

1. Grundlagen der Informatik

Zahlensysteme und interne Informationsdarstellung

Inhalt

- Grundlagen digitaler Systeme
- Boolesche Algebra / Aussagenlogik
- Organisation und Architektur von Rechnern
- Algorithmen, Darstellung von Algorithmen mit Struktogrammen und Programmablaufplänen
- **Zahlensysteme und interne Informationsdarstellung**

Zahlensysteme

Menschen nutzen zur Angabe von Werten und zum Rechnen vorzugsweise das **Dezimalsystem**

Beispiel

435 Fische aus dem Teich gefischt, d.h. $4 \cdot 10^2 + 3 \cdot 10^1 + 5 \cdot 10^0$

Digitale Rechner speichern Daten im **Dualsystem**

$435_{\text{dez}} = 1100110011_{\text{binär}}$

1100110011 bedeutet $1 \cdot 2^9 + 1 \cdot 2^8 + 1 \cdot 2^5 + 1 \cdot 2^4 + 1 \cdot 2^1 + 1 \cdot 2^0$

Andere Zahlensysteme sind z.B. das **Oktalsystem** und das **Hexadezimalsystem**

Zahlensysteme

Dezimalsystem, Binärsystem, Oktalsystem und Hexadezimalsystem sind s.g. Positionssysteme

Basis: B (10,2,8,16)

Ziffern: 0 bis $B-1$

Ziffern an Stellen von rechts nach links mit Faktor B^0 , B^1 , B^2 usw. bewertet

Es gibt Zahlensysteme, die keine Positionssysteme sind:
Römische Zahlenschrift, z.B. MMXIV für 2014

Zahlensysteme: Binärsystem

Binärdarstellung:

- ist die Interne Repräsentation der Daten, die vom Typ Integer und Double sind (inkl. vorzeichenlose und längere Formate)
- Werte werden bei Eingabe in Binärzahlen umgewandelt und intern so weiterverarbeitet
- einige Rechenoperationen und Vergleichsoperationen können auch eleganter und schneller programmiert werden, wenn man die binäre Repräsentation ausnutzt.

Binärdarstellung

Umrechnung Dezimal \rightarrow Binär

Eingabe: Dezimalzahl d

Ausgabe: Binärzahl auf Feld b
mit niederwertigster Stelle rechts

Prinzip: Teile Dezimalzahl wiederholt ganzzahlig durch 2 und notiere den Rest der Division von rechts nach links. Wiederhole solange wie die ganzzahlige Division noch ein Ergebnis ungleich Null liefert. Die notierten 0- und 1-Werte ergeben die Binärzahl.

Binärdarstellung

Umrechnung Dezimal → Binär

Beispiel: $d=35$

$35/2=17$, Rest = 1,	notiere	_ _ _ _ _	1
$17/2=8$, Rest = 1,	notiere	_ _ _ _	1 1
$8/2=4$, Rest = 0,	notiere	_ _ _	0 1 1
$4/2=2$, Rest = 0,	notiere	_ _	0 0 1 1
$2/2=1$, Rest = 0,	notiere	_	0 0 0 1 1
$1/2=0$, Rest = 1,	notiere		1 0 0 0 1 1

Binärdarstellung

Umrechnung Binär → Dezimal

- Per Wertigkeit der einzelnen Bits $b_n b_{n-1} \dots b_1 b_0$:
$$b_n * 2^n + b_{n-1} * 2^{n-1} + \dots + b_1 * 2^1 + b_0 * 2^0$$

Beispiel: $100011 \rightarrow 1*2^5 + 1*2^1 + 1*2^0 = 35$

- Durch Multiplizieren mit 2 und aufaddieren:
Binärzahl von links nach rechts durchlaufen, Summe (anfänglich 0) und Bitwert addieren.
Summe jedes mal mit 2 multiplizieren, außer bei letzter Bitstelle

Beispiel: 100011

$$((((((1*2+0)*2+0)*2+0)*2+1)*2)+1 = 35$$

Binar und Oktaldarstellung

Oktalzahlen werden anhand eines Positionssystems über 8 Ziffern gebildet:

Ziffern 0 bis 7

Zahl := $o_n o_{n-1} \dots o_1 o_0$

Der Wert der Zahl ergibt sich aus
 $o_n * 8^n + o_{n-1} * 8^{n-1} \dots o_1 * 8^1 + o_0 * 8^0$


Eine Oktalzahl kann aus der Binärzahl abgelesen werden und umgekehrt. Drei Binärstellen ergeben durch ihren Wert jeweils eine Stelle der entsprechenden Oktalzahl.

Die Gruppierung der Binärstellen muss von rechts nach links erfolgen (beginnend bei niederwertigster Stelle)


Binär- und Oktaldarstellung

Beispiele:

Binär 1 0 0 0 1 0 1 1 1 0
Oktal 1 0 5 6



Oktal 577
Binär 101 111 111



Binär- und Hexadezimaldarstellung

Hexadezimalzahlen werden anhand eines Positionssystems über 16 Ziffern gebildet:

Ziffern 0 bis F: 0 1 2 3 4 5 6 7 8 9 A B C D E F

Zahl := $h_n h_{n-1} \dots h_1 h_0$

Der Wert der Zahl ergibt sich aus

$$h_n * 16^n + h_{n-1} * 16^{n-1} \dots h_1 * 16^1 + h_0 * 16^0$$


Eine Hexadezimalzahl kann aus der Binärzahl abgelesen werden und umgekehrt. Vier Binärstellen ergeben durch ihren Wert jeweils eine Stelle der entsprechenden Hexadezimalzahl.

Die Gruppierung der Binärstellen muss von rechts nach links erfolgen (beginnend ab niederwertigster Stelle)


Binar und Hexadezimaldarstellung

Beispiele:

Binär 1 0 0 0 1 0 1 1 1 0
Hex 2 2 E



Hex 5FA
Binär 0101 1111 1010



Umwandlungen zwischen Positionssystemen (1)

Generalisierung:

Binär-, Oktal-, Dezimal- und Hexadezimalsysteme sind Positionssysteme mit unterschiedlicher Basis b :

$$\text{Wert} = z_n \cdot b^n + z_{n-1} \cdot b^{n-1} + \dots + z_1 \cdot b^1 + z_0 \cdot b^0$$

z_i sind jeweils Ziffern mit Wertevorrat im Intervall $[0, b-1]$

Zwischen zwei Systemen s_1 und s_2 ergibt sich eine Zusammenführbarkeit von Stellen wenn eine Basis eine ganzzahlige Potenz der anderen ist, d.h. es gilt

$$b_{s_2} = (b_{s_1})^n$$

n Ziffern des Systems s_1 ergeben eine Ziffer im System s_2

Umwandlungen zwischen Positionssystemen (2)

Gebräuchliche Umwandlungen:

s1	s2	n	
2	8	3	Binär ↔ Oktal
2	16	4	Binär ↔ Hexadezimal

Weitere mögliche Umwandlungen:

s1	s2	n	
2	4	2	Binär ↔ 4er System
4	16	2	4er ↔ Hexadezimal

Das Binärsystem wird benutzt, da man leicht zwei Werte in einer Speicherzelle speichern bzw. auf einer Leitung übertragen kann. Mittlerweile gibt's auch Speicherchips und Übertragungsverfahren, die mit mehrwertiger Logik arbeiten, z.B. MLC-Flash-Memories.

Datentypen

Obwohl alle Zahlen prinzipiell im Binärsystem dargestellt werden, ist bei der Programmierung eine Unterscheidung verschiedener Datentypen nötig:

Nichtnumerische Werte:

- Boolean (Wahrheitswerte)
- Zeichen und Zeichenketten (Strings)

Ausgewählte Bitmuster repräsentieren die Symbole für Buchstaben und Ziffern (siehe Kodierung)

Numerische Werte:

- Ganzzahlig
- Gebrochene Zahlen
- Varianten für verschiedene Wertebereiche und Genauigkeiten

Ganzzahlige Datentypen: unsigned integer

32-Bit: *unsigned integer*

Größe (rein binär)

Bit 31 30 ... ← ... 2 1 0

Der gespeicherte Zahlenwert errechnet sich aus den zugehörigen Zweierpotenzen:

$$b_{31}2^{31} + b_{30}2^{30} + \dots + b_22^2 + b_12^1 + b_02^0$$

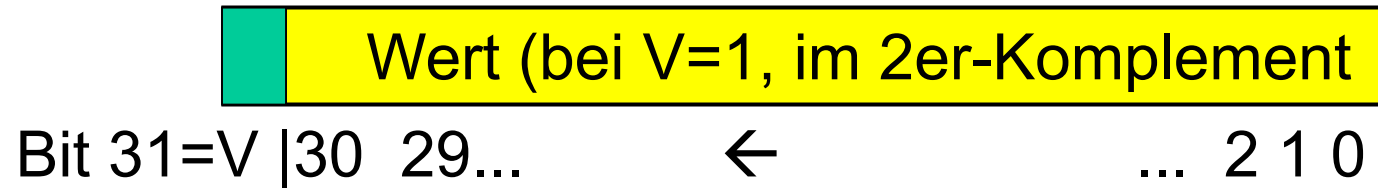
Die b_i stellen die betreffenden Bitwerte an der i -ten Stelle dar.

Maximaler Wert: $4294967295 = 2^{32} - 1$

Minimaler Wert: 0

Ganzzahlige Datentypen: integer

32-Bit: *signed integer*



Das werthöchste Bit (hier 31) ist das Vorzeichen-Bit V.

Die Größe errechnet sich *bei positiven Zahlen* aus den zugehörigen Zweierpotenzen:

$$b_{30}2^{30} + b_{29}2^{29} + \dots + b_22^2 + b_12^1 + b_02^0$$

Bei *negativen Zahlen* sind die b_i im sogenannten *Zweier-Komplement* dargestellt.

Ganzzahlige Datentypen: integer

Durch Aufteilung des Darstellungsbereichs in negative und positive Zahlen ändern sich der maximale und der minimale darstellbare Wert wie folgt:

Maximaler Wert: $2147483647 = 2^{31} - 1$

Minimaler Wert: $-2147483648 = -2^{31}$

Negative Zahlen im Zweierkomplement

Das Zweierkomplement einer Zahl erhält man, indem man die Binärdarstellung der Zahl bitweise komplementiert (Einerkomplement) und danach eine „1“ addiert.

Beispiel: Zahl = -1

Binärdarstellung: 0000 0000 0000 0001

Einerkomplement: 1111 1111 1111 1110

1 - Addition: **1111 1111 1111 1111**

Diese Darstellung ist die -1!

Man sieht, dass durch die Komplementierung immer das V-Bit auf „1“ gesetzt ist !

Negative Zahlen im Zweierkomplement

Eine Zahl im Zweierkomplement kann per Addition mit anderen Zahlen verknüpft werden.

Beispiel 1: $24 + (-7)$

$$24 = 00011000$$

$$7 = 00000111$$

$$-7 = 11111001$$

$$00011000$$

$$11111001$$

$$100010001 = 17 \text{ (ohne 1 links)}$$

Die Subtraktion kann auf eine Addition einer negierten Zahl abgebildet werden.

Negative Zahlen im Zweierkomplement

Eine Zahl im Zweierkomplement kann per Addition mit anderen Zahlen verknüpft werden.

Beispiel 2: $24 + (-28)$

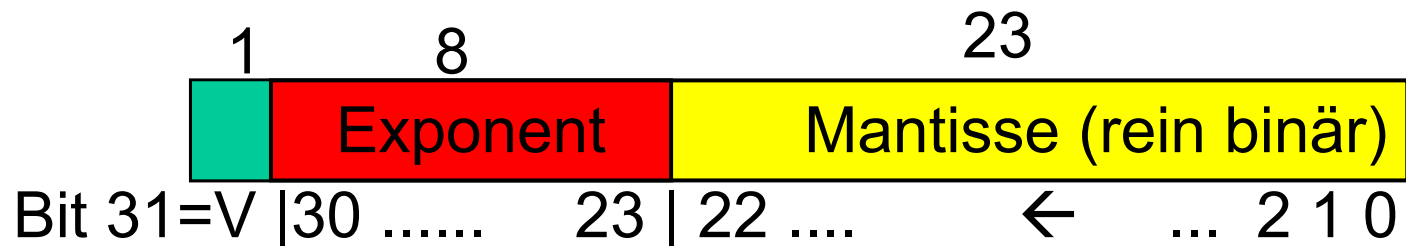
$$\begin{aligned} 24 &= 00011000 \\ 28 &= 00011100 \\ -28 &= 11100100 \end{aligned}$$

$$\begin{array}{r} 00011000 \\ 11100100 \\ \hline 01111100 \end{array} = -4 \quad (00000100 \text{ negiert und } +1)$$

Gleitkomma-Datentypen: single

Gleitkomma-Datentypen in verschiedenen Genauigkeitskategorien (*single*, *double*)

32-Bit: *single*



Das Vorzeichenbit V(Bit 31) bezieht sich nur auf den Wertanteil der Zahl. Der Wert der Mantisse m errechnet sich aus den zugehörigen **negativen** Zweierpotenzen:

$$m = b_{22}2^{-1} + b_{21}2^{-2} + \dots + b_22^{-21} + b_12^{-22} + b_02^{-23}$$

Die b_i stellen die betreffenden Bitwerte an der i -ten Stelle dar.

Gleitkomma-Datentypen: single

Zum Wert der Mantisse m wird immer eine 1 zusätzlich angenommen. Den Wert $1 + m$ nennt man **Signifikand**. Die 1 ist also nicht mit abgespeichert, sondern nur implizit vorhanden.

Der **Exponent** ist vorzeichenbehaftet und zur **Basis 2**. Dies wird erreicht, indem man den Exponenten um eine feste Größe (**BIAS**) erhöht darstellt. Der BIAS berechnet sich aus der Anzahl der Bits n des Exponenten:

$$\text{BIAS} = 2^{n-1} - 1$$

Mit $n=8$ ergibt sich 127.

Demnach muss der Binärwert des Exponenten e (die Bitstelle 23 hat dabei den Stellenwert 2^0) um den BIAS vermindert werden, um den realen Wert des Exponenten zur Basis 2 zu erhalten.

Gleitkomma-Datentypen: single

Der Wert Z einer 32-Bit Single-Zahl ist damit

$$Z = V \text{ Signifikand} * 2^{e - \text{BIAS}}$$

Das Vorzeichen V ist $+$, falls der Bitwert von $b_{31}=0$ ist, sonst $-$

Beispiel:

V Exponent

Mantisse

1 1000 0001 0100 0000 0000 0000 0000 000

- $e = 129$ $m = 0*2^{-1} + 1*2^{-2} + 0*2^{-3} + \dots + 0*2^{-23}$

- $e - \text{BIAS} = 2$ $m = 1*0.25 = 0.25$

$$Z = - (1+0.25) * 2^2 = - 1.25 * 4 = -5$$

Gleitkomma-Datentypen: single

Damit ergeben sich für **32-Bit Single**-Zahlen folgende Darstellungsbereiche:

Maximaler Wert: $3.40282e+38$

absolut minimaler Wert: $1.17549e-38$

(in dieser Darstellung ist e zur Basis 10 !)

Durch die begrenzte Anzahl von Bits zur Darstellung der Mantisse, ist die Anzahl genau darstellbarer Ziffern in einer Dezimaldarstellung ebenfalls begrenzt. Dabei werden die niederwertigen Ziffern betroffen.

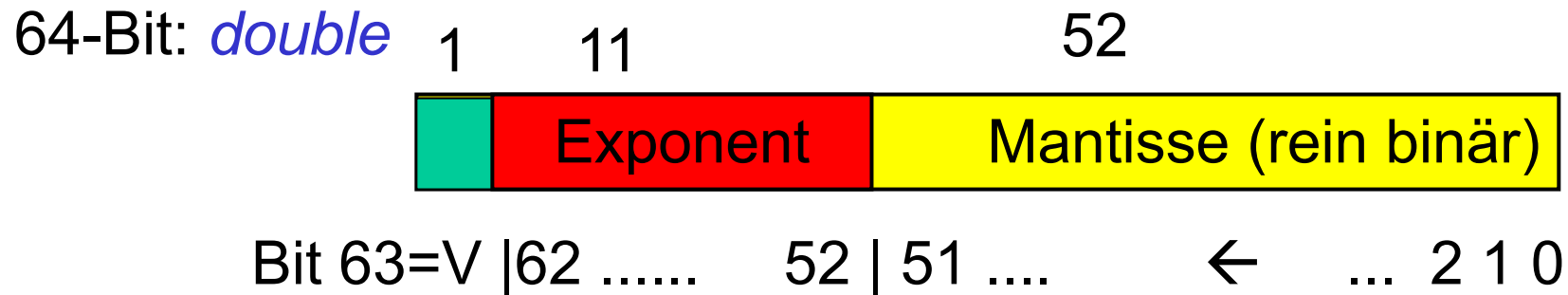
Anzahl der genau dargestellten Dezimalziffern: 6

Das heißt:

100100.8 und 100100.9 nicht mehr sicher unterscheidbar, da niederwertigste Ziffer sich an Position 7 befindet.

Bei 20 011 292 als Geldbetrag wären die Zehner- und Einerstelle nicht genau abgebildet.

Gleitkomma-Datentypen: double



Das Vorzeichenbit V(Bit 63) bezieht sich nur auf den Wertanteil der Zahl. Der Wert der Mantisse m errechnet sich aus den zugehörigen **negativen** Zweierpotenzen:

$$m = b_{51}2^{-1} + b_{50}2^{-2} + \dots + b_22^{-50} + b_12^{-51} + b_02^{-52}$$

Die b_i stellen die betreffenden Bitwerte an der i -ten Stelle dar.

Gleitkomma-Datentypen: double

Damit ergeben sich für **64-Bit double**-Zahlen folgende Darstellungsbereiche:

Maximaler Wert: $1.797693e+308$

absolut minimaler Wert: $2.225074e-308$

(in dieser Darstellung ist e zur Basis 10)

Anzahl der genau dargestellten Dezimalziffern: 15

Kodierung

Kodierung von Zeichen mit dem ASCII-Code

Weiterführende Aspekte zur Kodierung:

- Speicherplatzsparende Codes
- Fehlererkennende und -korrigierende Codes
- Verschlüsselnde Codes

Zeichen im ASCII-Code

Einzelne Zeichen, z.B. 'A', 'z', '7', oder '?' werden intern durch einen Zahlenwert repräsentiert, der binär kodiert wird.

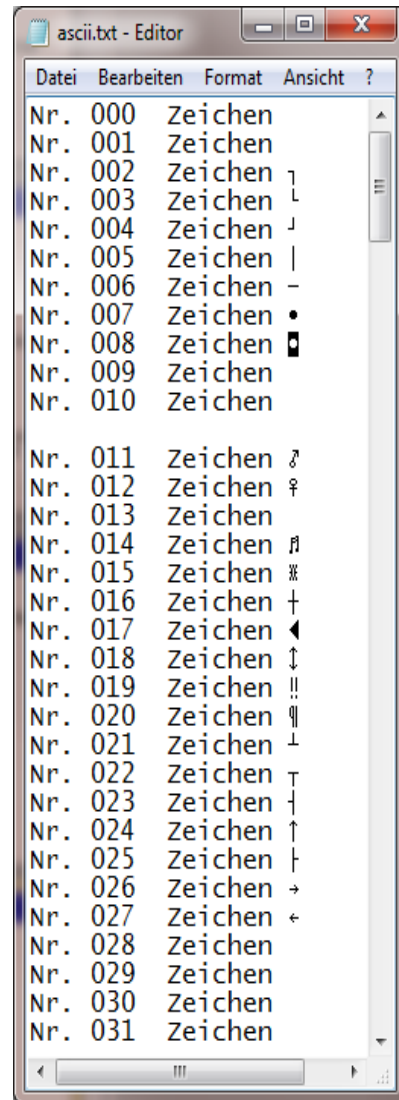
Ein Zeichen umfasst einen einzelnen Buchstaben, eine Ziffer, ein Sonderzeichen oder ein Steuerzeichen .

Es gibt verschiedene Möglichkeiten der Zuordnung des Zeichens auf den Zahlenwert

- Zuordnung muss eineindeutig sein
- Operationen (z.B. Vergleich) sollten sich einfach realisieren lassen

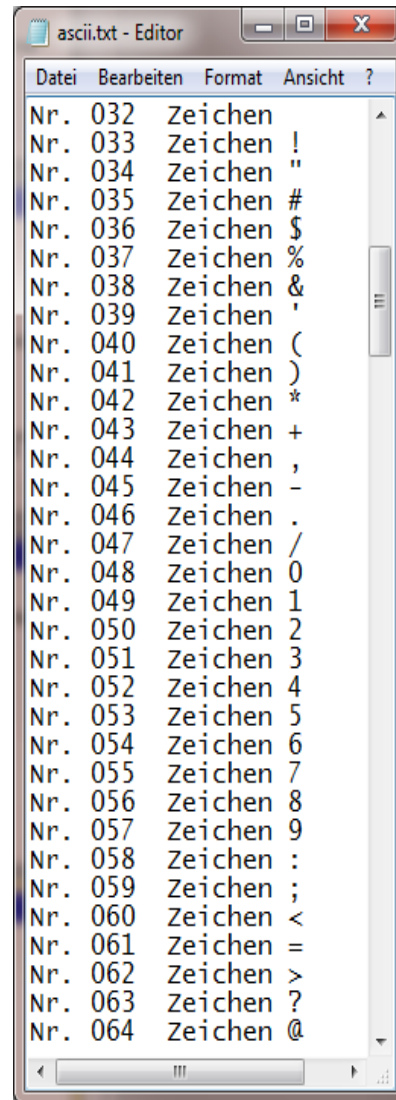
Eine spezielle Zuordnung ist dann ein Code, z.B. EBCDIC oder ASCII

ASCII-Code

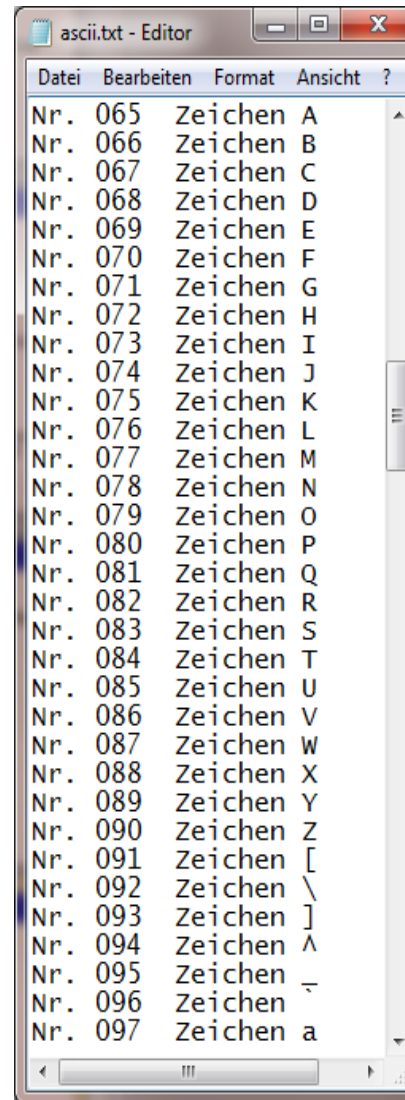


```
ascii.txt - Editor
Datei Bearbeiten Format Ansicht ?
Nr. 000 Zeichen
Nr. 001 Zeichen
Nr. 002 Zeichen 
Nr. 003 Zeichen 
Nr. 004 Zeichen 
Nr. 005 Zeichen 
Nr. 006 Zeichen 
Nr. 007 Zeichen 
Nr. 008 Zeichen 
Nr. 009 Zeichen 
Nr. 010 Zeichen

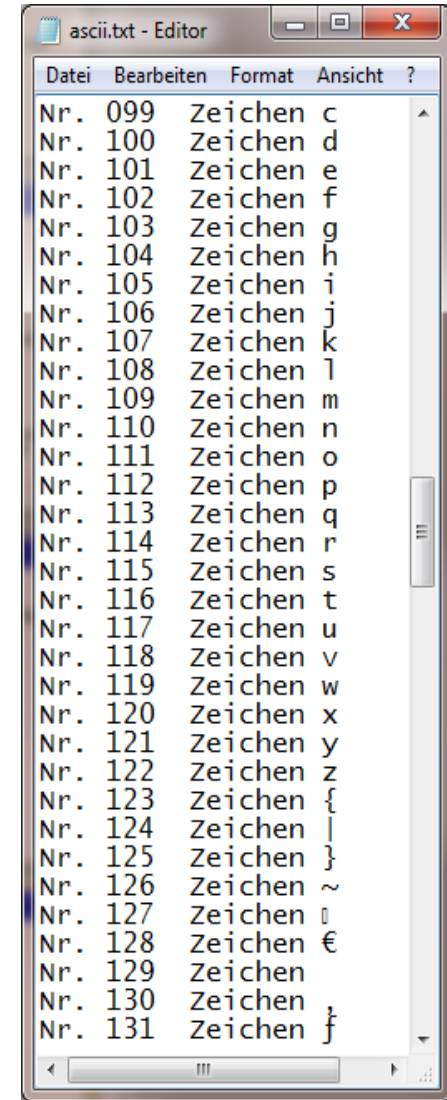
Nr. 011 Zeichen 
Nr. 012 Zeichen 
Nr. 013 Zeichen 
Nr. 014 Zeichen 
Nr. 015 Zeichen 
Nr. 016 Zeichen 
Nr. 017 Zeichen 
Nr. 018 Zeichen 
Nr. 019 Zeichen 
Nr. 020 Zeichen 
Nr. 021 Zeichen 
Nr. 022 Zeichen 
Nr. 023 Zeichen 
Nr. 024 Zeichen 
Nr. 025 Zeichen 
Nr. 026 Zeichen 
Nr. 027 Zeichen 
Nr. 028 Zeichen
Nr. 029 Zeichen
Nr. 030 Zeichen
Nr. 031 Zeichen
```



```
ascii.txt - Editor
Datei Bearbeiten Format Ansicht ?
Nr. 032 Zeichen
Nr. 033 Zeichen !
Nr. 034 Zeichen "
Nr. 035 Zeichen #
Nr. 036 Zeichen $
Nr. 037 Zeichen %
Nr. 038 Zeichen &
Nr. 039 Zeichen '
Nr. 040 Zeichen (
Nr. 041 Zeichen )
Nr. 042 Zeichen *
Nr. 043 Zeichen +
Nr. 044 Zeichen ,
Nr. 045 Zeichen -
Nr. 046 Zeichen .
Nr. 047 Zeichen /
Nr. 048 Zeichen 0
Nr. 049 Zeichen 1
Nr. 050 Zeichen 2
Nr. 051 Zeichen 3
Nr. 052 Zeichen 4
Nr. 053 Zeichen 5
Nr. 054 Zeichen 6
Nr. 055 Zeichen 7
Nr. 056 Zeichen 8
Nr. 057 Zeichen 9
Nr. 058 Zeichen :
Nr. 059 Zeichen ;
Nr. 060 Zeichen <
Nr. 061 Zeichen =
Nr. 062 Zeichen >
Nr. 063 Zeichen ?
Nr. 064 Zeichen @
```



```
ascii.txt - Editor
Datei Bearbeiten Format Ansicht ?
Nr. 065 Zeichen A
Nr. 066 Zeichen B
Nr. 067 Zeichen C
Nr. 068 Zeichen D
Nr. 069 Zeichen E
Nr. 070 Zeichen F
Nr. 071 Zeichen G
Nr. 072 Zeichen H
Nr. 073 Zeichen I
Nr. 074 Zeichen J
Nr. 075 Zeichen K
Nr. 076 Zeichen L
Nr. 077 Zeichen M
Nr. 078 Zeichen N
Nr. 079 Zeichen O
Nr. 080 Zeichen P
Nr. 081 Zeichen Q
Nr. 082 Zeichen R
Nr. 083 Zeichen S
Nr. 084 Zeichen T
Nr. 085 Zeichen U
Nr. 086 Zeichen V
Nr. 087 Zeichen W
Nr. 088 Zeichen X
Nr. 089 Zeichen Y
Nr. 090 Zeichen Z
Nr. 091 Zeichen [
Nr. 092 Zeichen \
Nr. 093 Zeichen ]
Nr. 094 Zeichen ^
Nr. 095 Zeichen _
Nr. 096 Zeichen `
Nr. 097 Zeichen a
```



```
ascii.txt - Editor
Datei Bearbeiten Format Ansicht ?
Nr. 099 Zeichen c
Nr. 100 Zeichen d
Nr. 101 Zeichen e
Nr. 102 Zeichen f
Nr. 103 Zeichen g
Nr. 104 Zeichen h
Nr. 105 Zeichen i
Nr. 106 Zeichen j
Nr. 107 Zeichen k
Nr. 108 Zeichen l
Nr. 109 Zeichen m
Nr. 110 Zeichen n
Nr. 111 Zeichen o
Nr. 112 Zeichen p
Nr. 113 Zeichen q
Nr. 114 Zeichen r
Nr. 115 Zeichen s
Nr. 116 Zeichen t
Nr. 117 Zeichen u
Nr. 118 Zeichen v
Nr. 119 Zeichen w
Nr. 120 Zeichen x
Nr. 121 Zeichen y
Nr. 122 Zeichen z
Nr. 123 Zeichen {
Nr. 124 Zeichen |
Nr. 125 Zeichen }
Nr. 126 Zeichen ~
Nr. 127 Zeichen 
Nr. 128 Zeichen 
Nr. 129 Zeichen
Nr. 130 Zeichen
Nr. 131 Zeichen 
```

Zeichen und Zeichenketten

ASCII ist ein 7-Bit-Code, 128 darstellbare Zeichen.

Später wurden 8 Bits ausgenutzt (ISO 8859-1). Die weiteren 128 Plätze wurden mit u.a. mit (westeuropäischen) sprachspezifischen Sonderzeichen belegt.

Unicode ist ein 2-Byte-Code, der die unteren 256 Plätze des ASCII-Codes übernommen hat, aber wesentlich mehr internationale Zeichen darstellen kann.

Unicode als 2-Byte-Code heute verbreitet

Zeichen und Zeichenketten

Zeichenketten sind Felder einzelner Zeichen, die zusammenhängende Texte aufnehmen.

Zeichenketten werden intern zusammenhängend gespeichert.

Jedes einzelne Zeichen wird durch den im Code zugewiesenen Zahlenwert repräsentiert

“Hallo“ besteht aus fünf aufeinanderfolgenden Zeichen: ‘H’, ‘a’, ‘l’, ‘l’, ‘o’

‘H’, ‘a’, ‘l’, ‘l’, ‘o’ wird repräsentiert durch 72, 97, 108, 108, 111

Programmiersprachen behandeln Zeichenketten unterschiedlich:

- Felder einzelner Zeichen (C)
- oder eigener Datentyp (String in C++, Java)

Speicherplatzsparende Codes

ASCII und Unicode: alle Zeichen werden mit einem s.g. Codewort gleicher Länge dargestellt.

Wenn die Zeichen mit unterschiedlicher Häufigkeit vorkommen, ist es möglich, die häufigsten mit einer kurzen Bitfolge und die seltenen mit einer längeren Bitfolge zu kodieren.

Man erreicht damit insgesamt eine Ersparnis an Speicherplatz (mittlere Anzahl Bits je Zeichen)

Beispiel: **Huffman-Code** für Zeichen 'A', 'B', 'C', 'D' unter der Annahme, dass nur diese 4 Zeichen in einer Nachricht vorkommen. Das Prinzip ist erweiterbar auf alle möglichen Zeichen.

Speicherplatzsparende Codes

Huffman-Code für Zeichen 'A', 'B', 'C', 'D'

Relative Häufigkeiten

'A':0.8, 'B':0.1, 'C':0.05, 'D':0.05

Code:

- 'A':0, 'B':10, 'C':110, 'D':111
- Häufigstes Zeichen wird mit kürzester Bitfolge kodiert
- Kein Codewort ist als Anfang in einem anderen Codewort enthalten

Zeichenfolge AAAABAAAACAAAAA (15 Zeichen) ergibt

000010000011000000 (18 Bits)

Fehlererkennende und –tolerierende Codes

Der bisher betrachtete Zeichencode (ASCII, Unicode) versagt, sobald ein Fehler bei der Speicherung, bzw. Übertragung auftritt.

Kodierungen für das Internet und Server-Speichersysteme können zum Teil Fehler erkennen und korrigieren.

Beispiele:

- **Einfacher Paritätscode:** Ein zusätzliches Bit, ein einzelner Bitfehler erkennbar, keine Korrektur möglich.
- **Hamming-Code**, Variante mit 4 Bits und 3 Paritätsbits (7,4), ein Fehler korrigierbar

Verschlüsselnde Codes

Originaldaten können bei Übertragung und Speicherung ver- und entschlüsselt werden.

Zusätzlich zur Zeichenkodierung werden die Codeworte (Bitfolgen) verändert (kryptografische Kodierung)

Einfachstes Verfahren: **One-Time-Pad**

- Veränderung erfolgt durch Summation der Codeworte mit geheimen Summanden (geheimer Schlüssel)
- Subtraktion des geheimen Anteils lässt Originaldaten wieder erscheinen
- Anwendung erfolgt bitweise, Summe und Differenz durch XOR (Exklusives ODER auf $\{0,1\}$ -Werten)
- Geheimer Anteil darf nicht wiederholt benutzt werden, Schlüssel muss so lang wie die Originalbitfolge sein

Verschlüsselnde Codes

Beispiel: **One-Time-Pad**

Originaltext: 'ABAACA'

Originaltext kodiert mit Huffman-Code:

'A': 0, 'B': 10, 'C': 110, 'D': 111,

010001100

Geheimer Schlüssel: 1011001101011....

Originaltext verschlüsselt: 111101010

Ergäbe 'DBBB'

Originaltext entschlüsselt mit geheimen Schlüssel 1011001101011...
ergibt 010001100, das ist 'ABAACA'