2 Bits and Bytes: Wie funktioniert ein Computer?

2.0.1. Jegliche Information (Texte, Musik, Videos) wird in Computern als Folge von **Bits** (= binary digits = Binärziffern) gespeichert, also durch eine Folge von Nullen und Einsen. Der Grund dafür ist, dass sich zwei Zustände (AN/AUS = ON/OFF = Spannung vorhanden/nicht vorhanden) elektronisch leicht realisieren lassen. Ein **Byte** ist (heute fast immer) eine Folge von 8 Bits.

2.1 Stellenwertsysteme allgemein

2.1.1. Es gibt viele Möglichkeiten, eine natürliche Zahl zu notieren. Hier sind drei Schreibweisen derselben Zahl:

- steinzeitlich (auch unäre Notation genannt): ●●●●●●●●●●●●●●●●●●●●●●●●
- römisch: XXIV
- heutzutage, im Dezimalsystem = Zehnersystem: 24

Unterschiedliche Notationen sind unterschiedlich gut zum Rechnen geeignet, weshalb es geschichtlich durchaus einen Unterschied gemacht hat (Effizienz in Handel und Verwaltung), welche Zahlenschreibweisen verwendet wurden bzw. überhaupt bekannt waren.

Heutzutage schreiben wir Zahlen meistens im Dezimalsystem, das heisst im *Stellenwertsystem zur Basis 10*. Die *indisch-arabischen Ziffern* 0, 1, …, 9, die wir verwenden, kann man bereits um etwa 300 v. Chr. in Indien nachweisen. Fibonacci lernte das Dezimalsystem in Algerien kennen und verbreitete es mit seinem Buch «Liber abaci» («Buch des Rechnens» und nicht «Buch des Abakus-Rechnens», vollendet 1202) in Italien und Europa. Erst im 15. Jahrhundert setzte es sich im deutschen Sprachraum gegen die römische Zahlschreibweise durch.

Statt der Basis 10 kann man beliebige andere natürliche Zahlen ≥ 2 als Basis nehmen.

Erklärung 2.1.2 Stellenwertsysteme, erste Beispiele

Stellenwertsysteme sind gewisse Notationssysteme für Zahlen, in denen Zahlen durch Ziffern dargestellt werden. Die *Stelle* (= Position) einer jeden Ziffer entscheidet über ihren *Wert*. Die Stellen werden von rechts her durchnummeriert, wobei die Stelle ganz rechts die Nummer 0 bekommt. Als **Basis** eines Stellenwertsystems kann man jede natürliche Zahl $b \geq 2$ nehmen. Die erlaubten Ziffern sind dann Symbole für die Zahlen von Null bis $b-1$.
Beispiele:

- **Stellenwertsystem zur Basis 10 = Dezimalsystem = Zehnersystem = 10er-System**
  Die erlaubten Ziffern sind 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 (beachte: $9 = 10-1$). Die Ziffer an der i-ten Stelle gibt an, wie oft die Zahl 10^i genommen wird.

Stelle 4	Stelle 3	Stelle 2	Stelle 1	Stelle 0
10000er= 10^4er	1000er= 10^3er	100er= 10^2er	10er= 10^1er	1er= 10^0er

- **Stellenwertsystem zur Basis 5 = Fünfersystem = 5er-System**
  Die erlaubten Ziffern sind 0, 1, 2, 3, 4 (beachte: $4 = 5-1$). Die Ziffer an der i-ten Stelle gibt an, wie oft die Zahl 5^i genommen wird.

Stelle 4	Stelle 3	Stelle 2	Stelle 1	Stelle 0
625er= 5^4er	125er= 5^3er	25er= 5^2er	5er= 5^1er	1er= 5^0er

2.1.3. Man beachte, dass die Null eine enorm wichtige Funktion in Stellenwertsystemen hat. Beispielsweise sind die Zahlen 2024 und 224 verschieden. Die Entstehung der Null als Zeichen für nichts ist historisch interessant.

Aufgabe 2.1 Kindergarten im Fünferland (im Fünferland wird das Fünfersystem verwendet)

(a) Schreibe alle Zahlen von 0 bis 31 im Fünfersystem auf (am besten rechtsbündig untereinander).

(b) Schreibe die aktuelle Jahreszahl im Fünfersystem.

Aufgabe 2.2 Primarschule im Fünferland

Notiere in der Tabelle links das «Kleine Eins-plus-Eins» und in der Tabelle rechts das «Kleine Ein-mal-Eins» im Fünferland. ✎

+					

·					

Aufgabe 2.3 Schriftliches Addieren funktioniert im 5er-System «genauso» wie im Zehnersystem. Man muss eigentlich nur aufpassen, dass man niemals eine der Ziffern 5, 6, 7, 8, 9 verwendet. Wer mag, kann die obige Additionstabelle «Kleines Eins-plus-Eins im 5er-System» verwenden.

Addiere schriftlich im 5er-System. Kontrolliere deine Rechnung anschliessend im Dezimalsystem.

a) (ohne Übertrag) $1423 + 2011$ b) (mit Überträgen) $1443 + 243$

Aufgabe 2.4 Schriftliches Multiplizieren funktioniert im 5er-System «genauso» wie im Zehnersystem. Wer mag, kann die obige Multiplikationstabelle «Kleines Ein-mal-Eins im 5er-System» verwenden.

Multipliziere schriftlich im 5er-System. Kontrolliere deine Rechnung anschliessend im Dezimalsystem.

a) $1402 \cdot 3$ b) $432 \cdot 123$

2.1.4. Wenn man Zahlen in verschiedenen Stellenwertsystemen notiert, sollte man angeben, welches Stellenwertsystem wo verwendet wird. Dies geschieht häufig durch Angabe der *im Dezimalsystem geschriebenen* Basis des Stellenwertsystems als rechtem unterem Index. Zum Beispiel gilt

$$2010_{10} = 31020_5$$

2.1.5. Wenn man eine Zahl im Zehnersystem schreibt, ist die Ziffer ganz rechts ihr Rest bei Division durch 10 und die verbleibenden Ziffern sind das Ergebnis der Ganzzahldivision. Zum Beispiel gilt

$$2024 : 10 = 202 \text{ Rest } 4$$

Die analoge Aussge gilt im Fünfersystem und motiviert den folgenden Algorithmus.

Algorithmus 2.1.6 Divisionsmethode: Umrechnen einer Zahl ins 5er-System (erklärt an einem Beispiel)

Schreibe die umzurechnende Zahl in die Box rechts oben (etwa die aktuelle Jahreszahl).
Dividiere diese Zahl (mit Rest) durch 5. Schreibe das Ergebnis (den sogenannten Ganzzahlquotient) in die Box links daneben und den Rest in die Box darunter.
Dividiere dann die neue Zahl in der ersten Zeile durch 5. Schreibe das Ergebnis in die Box links daneben und den Rest in die Box darunter. Wiederhole dies, bis beim Dividieren Null als Ergebnis (nicht als Rest) herauskommt.

Als Ergebnis erhält man in der unteren Zeile die gesuchte Darstellung im 5er-System.
In unserem Beispiel gilt:

Aufgabe 2.5 Verwende das Divisionsverfahren (Algorithmus 2.1.6), um die folgenden Dezimalzahlen im 5er-System darzustellen.

Hinweis: Wer im Kopf durch 5 dividieren möchte: Division durch 5 ist dasselbe wie Multiplikation mit 2 und Division durch 10, denn $\frac{1}{5} = \frac{2}{10}$.

a) 194_{10} b) 678_{10} c) 2930_{10}

Binär- und Hexadezimalsystem: Die beiden wichtigsten Stellenwertsysteme in der Informatik

Erklärung 2.1.7 Stellenwertsystem zur Basis 2 = Binärsystem = Dualsystem = 2er-System

Binärsystem:
Die erlaubten Ziffern sind 0, 1. Die Ziffer an der i-ten Stelle gibt an, wie oft die Zahl 2^i genommen wird.

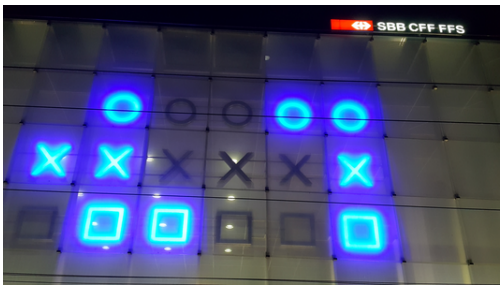
Stelle 4	Stelle 3	Stelle 2	Stelle 1	Stelle 0
16er= 2^4er	8er= 2^3er	4er= 2^2er	2er= 2^1er	1er= 2^0er

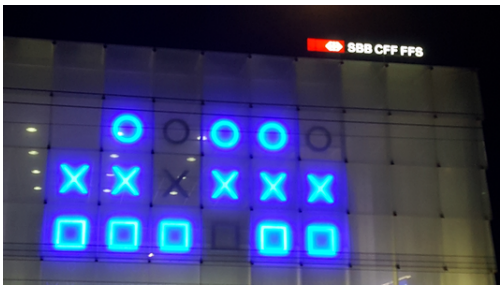
Schreibweise: Längere Binärzahlen werden in Vierer-Gruppen notiert: z.B. $1976_{10} = 111'1011'1000_2$. Grund dafür ist die übliche Gruppierung von 8 Bits in ein Byte, und die einfache Umrechnung ins 16er-System.

Aufgabe 2.6 Umrechnen vom Binärsystem ins Zehnersystem.

Bei der St. Galler Bahnhofsuhr wird die aktuelle Uhrzeit zeilenweise (Stunden, Minuten, Sekunden) im Binärsystem angegeben. Licht an = Ziffer 1; Licht aus = Ziffer 0. Die Form der Symbole (Kreis, Kreuz, Quadrat) dient nur der Unterscheidung von Stunden, Minuten und Sekunden.

Welche Uhrzeit wird jeweils angezeigt?

a)


b)


Aufgabe 2.7

(a) Kindergarten im 2erland: Zähle binär von 0 bis 33

(b) Primarschule im 2erland: Fülle die Eins-plus-Eins- und Einmal-Eins-Tabelle rechts aus.

+		

·		

Aufgabe 2.8 Addiere schriftlich im Binärsystem. Kontrolliere deine Rechnung anschliessend im Dezimalsystem.

a) $10011 + 1111$ b) $11111 + 11111$

Aufgabe 2.9 Schriftliches Multiplizieren im Binärsystem ist super-einfach. Multipliziere schriftlich und kontrolliere deine Rechnung anschliessend im Dezimalsystem.

a) $10011 \cdot 11001$ b) $11111 \cdot 11111$

Aufgabe 2.10 (Das Zahnradsymbol bedeutet «Bonus-Aufgabe»)

Schriftliches Dividieren (mit Rest oder mit Nachkommastellen) funktioniert in jedem Stellenwertsystem.

- Dividiere schriftlich im Binärsystem und kontrolliere dein Resultat im Dezimalsystem:

a) 11110000 : 10 b) 111100 : 101 c) (mit Rest) 111111 : 101

- Es gibt auch binäre Kommazahlen. Die Nachkommastellen im Binärsystem stehen für $2^{-1} = \frac{1}{2}$, $2^{-2} = \frac{1}{4}$, etc. Berechne $1011 : 101 = \frac{1011}{101}$ als Kommazahl.

Erklärung 2.1.8 Stellenwertsystem zur Basis 16 = Hexadezimalsystem = 16er-System

Hexadezimalsystem: Die erlaubten Ziffern sind 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F. Dabei steht «A» für zehn, «B» für elf, ..., «E» für vierzehn und «F» für fünfzehn. Die Ziffer an der i-ten Stelle gibt an, wie oft die Zahl 16^i genommen wird.

Stelle 4	Stelle 3	Stelle 2	Stelle 1	Stelle 0
65536er= 16^4er	**4096er**= 16^3er	**256er**= 16^2er	**16er**= 16^1er	**1er**= 16^0er

Aufgabe 2.11 Kindergarten im 16erland: Zähle hexadezimal von 0 bis 33.

Aufgabe 2.12 Stelle die folgenden (im Hexadezimalsystem angegebenen Zahlen) im Dezimalsystem dar.

a) FF_{16} b) AFFE_{16} c) $\mathrm{CAFE07}_{16}$

2.1.9. Das Divisionsverfahren (Algorithmus 2.1.6) funktioniert sinngemäss für jedes Stellenwertsystem. Beispielsweise muss man beim Umrechnen ins Binärsystem Division durch 2 und beim Umrechnen ins Hexadezimalsystem Division durch 16 verwenden.

Aufgabe 2.13 Wandle die folgenden Dezimalzahlen mit Hilfe des Divisionsverfahrens (Algorithmus 2.1.6) sowohl ins Binär- als auch ins Hexadezimalsystem um.

a) 2024_{10} b) 3386_{10}

Algorithmus 2.1.10 Umwandlung zwischen Binär- und Hexadezimalsystem und umgekehrt

Will man eine Binärzahl in eine Hexadezimalzahl umwandeln, so schreibt man ihre Ziffern in die Quadrate in der oberen Zeile der folgenden «Tabelle» (so, dass die Ziffer ganz rechts im Quadrat ganz rechts steht). Nun wandelt man jeden «Vierer-Block» von Binärziffern in die zugehörige (einstellige) Hexadezimalzahl um und schreibt diese in das Rechteck darunter. Dann steht in der unteren Zeile die gesuchte Hexadezimalzahl.

In unserem Beispiel gilt also

Die Umwandlung vom Hexadezimalsystem ins Binärsystem geht «umgekehrt»: Aus jeder Hexadezimalziffer wird ein Block von vier Binärziffern.

Dieses Verfahren funktioniert wegen $2^4 = 16$.

Merke 2.1.11 Binär- und Hexadezimalzahlen in Python

In Python (und vielen anderen Programmiersprachen) können auch direkt binäre und Hexadezimale Zahlen mit den Präfixen 0b und 0x eingegeben werden. Als Trennzeichen werden Bodenstriche verwendet.

```
d = 42          # Dezimalzahl, im Speicher dann sowieso binär
e = 0b10_1010   # Binärzahl, ebenfalls 42
f = 0x2A        # Hexadezimalzahl, ebenfalls 42
```

Aufgabe 2.14 Konvertiere mit dem Algorithmus 2.1.10 die folgenden Zahlen vom Binärsystem ins Hexadezimalsystem bzw. umgekehrt.

a) 0b1111_0000_0101 b) 0b11_0011_1101 c) 0xFF d) 0xC_AFE07

2.1.12. Computer rechnen mit Binärzahlen. Ein Nachteil von Binärzahlen ist, dass relativ kleine Zahlen bereits relativ viele Stellen benötigen; zum Beispiel hat die Dezimalzahl 2048 binär bereits zwölf Stellen, $2048_{10} = 1000'0000'0000_2$. Hexadezimalzahlen benötigen hingegen nur etwa ein Viertel der Stellen, z. B. gilt $1000'0000'0000_2 = 800_{16}$. Da die Umrechnung zwischen Binär- und Hexadezimalzahlen sehr einfach ist (und deutlich einfacher als die Umrechnung ins Dezimalsystem), werden Hexadezimalzahlen im informatischen Kontext oft verwendet (etwa bei der Kodierung von Farben, siehe später).

Aufgabe 2.15 Schreiben Sie in Python ein Programm, das eine (Dezimal-)Zahl im Binär- und im Hexadezimalsystem ausgibt (bzw. in einem Stellenwertsystem deiner Wahl).

Empfehlung: Implementiere den (angepassten) Algorithmus 2.1.6 als Funktion `konvertiere(x, b)`, die die Zahl `x` ins Stellenwertsystem mit der Basis `b` konvertiert und das Ergebnis als String zurückgibt.

```
def konvertiere(x,b):
    zahl = ""
    while ?????:
        # Tu wat und berechne ziffer
        # Ziffer in Zeichenkette umwandeln und einfügen
        zahl = str(ziffer) + zahl
    return zahl

# Testen
print(konvertiere(42,2))      # 1010101
print(konvertiere(3267,7))    # 12345
```

Wer allgemeiner Basen über 10 (bis maximal 36) mag, kann eine Ziffer wie folgt in eine Zeichenkette umrechnen:

```
if ziffer<10:
    ziffer = str(ziffer)  # Dezimalziffer, einfach umwandeln
else:
    # Code vom Buchstaben A ermitteln (mit ord),
    # addieren, mit chr zurück in Symbol umwandeln
    ziffer = chr(ord('A')+ziffer-10)
```

Und warum ist die Zahl $3'735'928'559_{10}$ im Hexadezimalsystem für Vegetarier ungeeignet?

2.1.13. Etwas seltener wird in der Informatik auch das Oktalsystem (d.h. Zahlensystem mit Basis 8) verwendet. Insbesondere dort, wo Information in 3-Bit Gruppen vorliegt.

Warum verwechseln Informatiker immer wieder das Datum von Halloween mit jenem von Weihnachten?

Antwort:

0x4f 0x43 0x54 0x20 0x33 0x31 0x20 0x3d 0x20 0x44 0x45 0x43 0x20 0x32 0x35

2.2 Speicherung von Daten am Computer allgemein

2.2.1. Computer können nur Nullen und Einsen abspeichern. Alle Daten (Zahlen, Texte, Bilder, Musik, Videos, Programme, Computerspiele) müssen deshalb in geeignete Folgen von Nullen und Einsen umgewandelt werden.

Dass alle digitale Information in Einsen und Nullen vorliegt, stimmt fast uneingeschränkt, wenn man nur die Softwareseite betrachtet.

Flashspeicher arbeiten heute aber tatsächlich mit zu 16 Ladungszuständen, speichern also bis zu 4 Bits in einer Speicherzelle.

Gigabit Ethernet arbeitet mit 5 Spannungszuständen, wobei die nicht einfach nur codierend sind, sondern immer auch noch geschickt wechseln müssen, um eine stabile Signalübertragung zu gewährleisten.

Merke 2.2.2

Jede Datei bzw. auf dem Computer gespeicherte Information wird durch eine Folge von Nullen und Einsen codiert, die als eine (sehr grosse) natürliche Zahl im Binärsystem aufgefasst werden kann!

2.3 Speicherung von Text

2.3.1. Jeder Text, der am Computer abgespeichert werden soll, muss in eine Folge von Nullen und Einsen (= eine Binärzahl) umgewandelt werden. Naheliegend ist die Idee, jedem Zeichen eine Zahl zuzuordnen und diese als Binärzahl abzuspeichern.

Allgemein nennt man eine Zuordnung von gewissen Zeichen (aus einem «Alphabet») zu gewissen anderen Dingen (etwa Zahlen) eine «Kodierung» oder kurz einen «Code».

Ein wichtiges Beispiel einer Kodierung ist der Morse-Code; bekannt ist vermutlich die Morse-Gruppe «drei kurz, drei lang, drei kurz» für «SOS». Beim Morsen wird jedem Buchstaben eine Folge langer und kurzer Töne zugeordnet; unterschiedliche lange Pausen dienen der Abgrenzung von Tönen, übermittelten Buchstaben und Wörtern.

Aufgabe 2.16 Wie würden Sie folgenden Morse-Code mit aussliesslich Nullen und Einsen codieren?

```
--. .- .-. / -. .. -.-. .... - / ... --- / . .. -. ..-. .- -.-. ....
```

ASCII-Code

2.3.2. Zur Speicherung von (englischsprachigen) Texten auf Computern hat sich der sogenannte ASCII-Code durchgesetzt (*American Standard Code for Information Interchange*; erste Version von 1963, aktuelle Version von 1968). Er ordnet jedem Zeichen eine 7-stellige Binärzahl zu. Die Codierung geht aus der folgenden vierspaltigen ASCII-Tabelle.

Quelle: https://github.com/stevenlinx/Four-Column-ASCII

Vom Zeichen zum ASCII-Code: Das Zeichen F findet sich in der Spalte 10 und der Zeile 00110; hintereinandergeschrieben ergibt sich der zugehörige ASCII-Code 10 00110 = $(1000110)_2 = (46)_{16} = (70)_{10}$.

Der ASCII-Code eines Zeichens ist eine Zahl, die man in jedem beliebigen Stellenwertsystem angeben kann.

Vom ASCII-Code zum Zeichen: Gegeben ist der ASCII-Code $(43)_{10} = (2B)_{16}$ (als Dezimal- bzw. Hexadezimalzahl). Wandle diese Dezimalzahl in eine **siebenstellige** (mit führenden Nullen) Binärzahl um: $(43)_{10} = (2B)_{16} = (0101011)_2$. Ihre ersten beiden Stellen 01 liefern die Spalte und die hinteren fünf Stellen 01011 ihre Zeile in der Tabelle rechts: Das zugehörige Zeichen ist das Pluszeichen.

Steuerzeichen: Die Zeichen in der Zeile 00 sind sogenannte Steuerzeichen (control characters); beispielsweise steht LF für *line feed* (Zeilenvorschub; neue Zeile); CR steht für *carriage return* (Wagenrücklauf, etwa zum Steuern von Druckern). Auch DEL (ganz rechts unten) ist ein Steuerzeichen, es steht für *delete*.

Druckbare Zeichen: Die anderen 128 - 32 - 1 = 95 Zeichen sind druckbare Zeichen (Spc steht für *space* (Leerzeichen)).

ASCII ist (ursprünglich) eine 7-Bit-Zeichenkodierung: Der ASCII-Code ist eine 7-Bit-Zeichenkodierung, der $2^7 = 128$ Zeichen (davon 95 druckbar) codiert, die sogenannten ASCII-Zeichen. Da Byte (= 8 Bit) die übliche Speichereinheit ist, wird meist ein Byte zum Speichern eines mit ASCII kodierten Zeichens verwendet.
Vielleicht wurde bemerkt, dass im klassischen 7-Bit-ASCII einige Zeichen fehlen (etwa die Umlaute ä, ö, ü, das Euro-Zeichen). Einige dieser Zeichen wurden später in gewissen 8-Bit-Erweiterungen von ASCII aufgenommen.

00	01	10	11	
NUL	Spc	@	`	00000
SOH	!	A	a	00001
STX	"	B	b	00010
ETX	#	C	c	00011
EOT	$	D	d	00100
ENQ	%	E	e	00101
ACK	&	F	f	00110
BEL	'	G	g	00111
BS	(	H	h	01000
TAB	)	I	i	01001
LF	*	J	j	01010
VT	+	K	k	01011
FF	,	L	L	01100
CR	-	M	m	01101
SO	.	N	n	01110
SI	/	O	o	01111
DLE	0	P	p	10000
DC1	1	Q	q	10001
DC2	2	R	r	10010
DC3	3	S	s	10011
DC4	4	T	t	10100
NAK	5	U	u	10101
SYN	6	V	v	10110
ETB	7	W	w	10111
CAN	8	X	x	11000
EM	9	Y	y	11001
SUB	:	Z	z	11010
ESC	;	[	{	11011
FS	<	\	|	11100
GS	=	]	}	11101
RS	>	^	~	11110
US	?	_	DEL	11111

Aufgabe 2.17

(a) Welche Zeichenfolge (die im Wesentlichen so auf der Festplatt eines Computers stehen könnte) codiert die folgende Folge von (8-stelligen) Binärzahlen (= Bytes)?

0100'0001 0110'1100 0110'1100 0110'0101 0111'0011 0010'0000 0110'1001 0111'0011 0111'0100
0010'0000 0100'0010 0110'1001 0110'1110 0110'0001 0110'0101 0111'0010 0111'1010 0110'0001
0110'1000 0110'1100 0010'0001

Hinweis: Die erste Ziffer jeder Binärzahl ist Null und kann ignoriert werden. Die verbleibende 7-stellige Binärzahl ist mit der ASCII-Tabelle zu dekodieren.

(b) Welche Zeichenfolge codiert die folgende Folge von (zweistelligen) Hexadezimalzahlen (= Bytes)?

47 75 74 20 67 65 6D 61 63 68 74 21

Hinweis: Jede zweistellige Hexadezimalzahl ist in eine 8-stellige Binärzahl umzuwandeln, welche dann per ASCII-Tabelle für ein Zeichen steht.

(c) (freiwillig) Wer mag, kann etwa in Visual Studio Code einen Hex-Editor (= Hexadezimal-Editor) als Erweiterung/Extension installieren und sich eine beliebige Datei oder die gerade entschlüsselten Texte damit anschauen (Rechtsklick auf Datei, «Open/Reopen Editor With ...»).

	00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F	Decoded Text
00000000	57 69 65 20 73 70 65 69 63 68 65 72 74 20 65 69	Wie speichert ei
00000010	6E 20 43 6F 6D 70 75 74 65 72 20 64 69 65 73 65	n Computer diese
00000020	6E 20 54 65 78 74 20 61 6C 73 20 46 6F 6C 67 65	n Text als Folge
00000030	20 76 6F 6E 20 48 65 78 61 64 65 7A 69 6D 61 6C	von Hexadezimal
00000040	2D 5A 61 68 6C 65 6E 20 3D 20 42 79 74 65 73 3F	-Zahlen = Bytes?

Abbildung 1: Text-Datei im Hex-Editor; die Hexadezimalzahlen links entsprechen genau den Zeichen rechts. Die Tabelle links ist im Wesentlichen ein «Blick auf die Festplatte» wo die Hexadezimalzahlen als Binärzahlen stehen.

Unicode

Für verschiedene Sprachen und schriftsysteme wurden eigene Codierungen verwendet, z.T. Erweiterungen von ASCII mit den verbbleibenden 128 ungenutzten Werten, z.T. gänzlich eigene Codierungen.

Die Konvertierung zwischen diesen Codierungen erfolgt nicht immer erfolgreich und man hat oft Texte erhalten, die unsinnige Zeichen anstatt Umlaute enthielten, wie z.B. Bl&x#chliger anstatt Blöchliger.

Um dem Chaos ein Ende zu bereiten, hat sich das «Unicode Consortium» zum Ziel gesetzt, sämtliche Schriftsprachen mit einer Codierung darstellen zu können. Mit diesem Ziel wurde der «Unicode» geschaffen, wo heute 155,063 Schriftzeichen definiert sind. Damit sind 168 Schriftsprachen und diverse andere Dinge wie mathematische oder musikalische Notation, aber auch Emoties definiert.

Fast alle Computer-Systeme verwenden heute Unicode. Die Frage ist, wie diese Zahlen dann effektiv codiert werden sollen. Dazu gibt es verschiede Standards. Eine kaum verwendete Methode verwendet einfach 4 Bytes pro Buchstabe, was aber viel Platz verbraucht. Die meist verwendete Codierung ist UTF-8, die für ASCII-Zeichen nur ein Byte benötigt und für andere Zeichen 2 bis 5 Bytes. Die Codierung ist de facto Standard auf dem Web und den meisten Systemen, abgesehen von Windows.

Aufgabe 2.18 Konvertieren Sie zwischen Unicode und Zeichen:

a) A b) ö c) 128’169 d) 9’835

Aufgabe 2.19 Studieren Sie den Wikipedia-Artikel zur UTF-8 Codierung und codieren Sie die Unicode-Zeichen der letzten Aufgabe in UTF-8 binär. Heben Sie dabei die Codierenden Bits hervor.

Speicherplatzbedarf

2.3.3 (Erinnerung: Bits and Bytes)**.**

- **Bit** = **b**inary digi**t** = Binärziffer, also 0 oder 1.
- **Byte** = Folge von 8 Bit = 8-stellige Binärzahl, z. B. 0100 1101 bzw. (im Kontext von Speichern, etwa Festplatten) die Möglichkeit, eine solche Zahl zu speichern.

Weil man an jeder der 8 Positionen zwei mögliche Ziffern hat, kann ein Byte $2^8 = 256$ verschiedene Werte annehmen, nämlich alle Binärzahlen von 0b0000’0000 bis 0b1111’1111.

2.3.4. Vermutlich ist bekannt, dass

- 1 Kilobyte, 1 kB, für $1'000 = 10^3$ Byte steht;
- 1 Megabyte, 1 MB, für $1'000'000 = 10^6$ Byte steht;
- 1 Gigabyte, 1 GB, für $1'000'000'000 = 10^9$ Byte steht;
- 1 Terabyte, 1 TB, für $1'000'000'000'000 = 10^{12}$ Byte steht.

Es gibt neben Dezimalpräfixen auch auch «Binärpräfixe». Beispiele: 1 Kibi-Byte = 1 KiB = $2^{10} = 1024$ Byte; 1 Mebi-Byte = 1 MiB = $2^{20} = 1'048'576$ Byte.

Aufgabe 2.20 Auf dem Computer sind mit kB, MB und GB normalerweise immer die Binärpräfixe gemeint. Speichermedienhersteller verwenden aber immer Dezimalrpäfixe. Warum wohl?

Sie kaufen eine externe Harddisk mit 2TB Kapazität. Sie schliessen die Harddisk zu Hause an. Wie viele TB Kapazität wird das System in etwa anzeigen? Warum auf jeden Fall noch etwas weniger?

Aufgabe 2.21

(a) Auf eine Seite DIN-A4-Papier passen ca. 2000 Text-Zeichen (in normaler Grösse). Wenn man jedes Zeichen per ASCII durch ein Byte codiert, wieviel Speicherplatz benötigt man, um den Inhalt von 500 Seiten Text abzuspeichern?

(b) Wie viele solche 500-seitigen Bücher (die nur aus Text bestehen) kann man auf einer handelsüblichen 400 Gigabyte-Festplatte abspeichern?

(c) Die grösste Biblothek der Welt ist die British Library mit ca. 14 Millionen Büchern (unter etwa 200 Millionen Medieneinheiten, laut englischer Wikipedia vom 17.01.2022). Wenn man vereinfachend annimmt, dass ein Buch durchschnittlich 500 Seiten hat und nur aus Text besteht, wie viele handelsübliche Laptops mit 400 Gigbyte-Festplatten benötigt man in etwa, um diese 14 Millionen Bücher abzuspeichern?

2.4 Lösungen

Hinweise zu den Symbolen:

⚒ Diese Aufgaben könnten (mit kleinen Anpassungen) an einer Prüfung vorkommen. Für die Prüfungsvorbereitung gilt: "If you want to nail it, you'll need it".

✸ Diese Aufgaben sind wichtig, um das Verständnis des Prüfungsstoffs zu vertiefen. Die Aufgaben sind in der Form aber eher nicht geeignet für eine Prüfung (zu grosser Umfang, nötige «Tricks», zu offene Aufgabenstellung, etc.). **Teile solcher Aufgaben können aber durchaus in einer Prüfung vorkommen!**.

Diese Aufgaben sind dazu da, über den Tellerrand hinaus zu schauen und/oder die Theorie in einen grösseren Kontext zu stellen.

⚒ Lösung zu 2.1 ex-kindergarten

(a) 0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24, 30, 31, 32, 33, 34, 40, 41, 42, 43, 44, 100, 101, 102, 103, 104, 110, 111

(b) $2025_{10} = 31100_5$

⚒ Lösung zu 2.2 ex-primarschule-5erland

+	0	1	2	3	4
0	0	1	2	3	4
1	1	2	3	4	10
2	2	3	4	10	11
3	3	4	10	11	12
4	4	10	11	12	13

·					
	0	0	0	0	0
	0	1	2	3	4
	0	2	4	11	13
	0	3	11	14	22
	0	4	13	22	31

⚒ Lösung zu 2.3 ex-schriftlich-addieren-5er

a)

	5er	*10er*
	1423	238
+	2011	256
	3434	494

b)

	5er	*10er*
	1443	248
+	243	73
	2241	321

⚒ Lösung zu 2.4 ex-schriftlich-multiplizieren-5er

⚒ Lösung zu 2.5 ex-umrechnen

⚒ Lösung zu 2.6 ex-bahnhofsuhr-ablesen

⚒ Lösung zu 2.7 ex-kindergarten-2

⚒ Lösung zu 2.8 ex-schriftlich-addieren

⚒ Lösung zu 2.9 ex-schriftlich-multiplizieren

Lösung zu 2.10 ex-schriftlich-dividieren

Lösung zu 2.11 ex-kindergarten-16

Lösung zu 2.12 ex-umrechnen-hexadezimal-dezimal

Lösung zu 2.13 ex-umrechnen-nach-binaer-und-hexadezimal

Lösung zu 2.14 ex-binaer-hexadezimal

a) 0xF05 b) 0x33D c) 0b1111_1111
d) 0b1100_1010_1111_1110_0000_0111

Lösung zu 2.15 ex-umrechnen-python

```
def konvertiere(x,b):
    if x==0:
       return "0"
    zahl = ""
    while x>0:
        ziffer = x % b     # Rest
        x = x // b         # Ganzzahldivision
        if ziffer < 10:
            ziffer = str(ziffer)
        else:
            ziffer = chr(ord('A')+ziffer-10)
        zahl = ziffer + zahl
    return zahl

# Testen
print(konvertiere(42,2))              # 1010101
print(konvertiere(3267,7))            # 12345
print(konvertiere(3735928559, 16))    # DEADBEEF
```

Lösung zu 2.16 ex-morsecode-in-binaer

Der Morsecode steht für «gar nicht so einfach».

Einfach Strich und Punkt zu Eins und Null (oder umgekehrt) zu übersetzen, funktioniert nicht, weil damit weder die Abstände zwischen den Buchstaben, noch jene zwischen den Wörtern codiert werden können.

Damit haben wir effektiv 4 Symbole (Punkt, Strich, Abstand, Worttrenner), die codiert werden müssen. Das kann mit 2 Bits geschehen, die zusammen 4 Zustände annehmen können, z.B. wie folgt:

00: Punkt
01: Strich
10: Buchstabe fertig
11: Wort fertig

Lösung zu 2.17 ex-ascii-entschluesseln

(a) Alles ist Binaerzahl!
(b) Gut gemacht!

Lösung zu 2.18 ex-unicode-finden

a) 65
b) 0xf6 = 246

c) Pile of poo

d) 2 verbundene Achtelnoten

Lösung zu 2.19 ex-utf8-codieren

Lösung zu 2.20 ex-dec-vs-bin

Für die gleiche Datenmenge, erhält man mit dezimalen Präfixen eine grössere Masszahl als mit binären Präfixen. Das verkauft sich besser.

Ein TiB sind $2^{40} = 1'099'511'627'776$ Bytes. D.h. eine Harddisk mit $2 \cdot 10^{12}$ Bytes enthält also $2 \cdot 10^{12}/2^{40} \approx 1.819$ TiB.

Weil zusätzlich Verwaltungsinformation gespeichert werden muss (Dateinamen, etc.) wird die effektiv nutzbare Kapazität noch ein bisschen kleiner sein.

Lösung zu 2.21 ex-buecher-auf-festplatte

(a) Pro Seite benötigt man 2000 Byte = 2 Kilobyte. Für 500 Seiten benötigt man also $500 \cdot 2$ Kilobyte = 1000 Kilobyte = 1 Megabyte.

(b) Wegen 400 Gigabyte = 400'000 Megabyte kann man 400'000 Bücher darauf abspeichern.

(c) Pro Laptop kann man 400'000 Bücher speichern, man benötigt also etwa $\frac{14'000'000}{400'000} = \frac{140}{4} = 35$ Laptops.