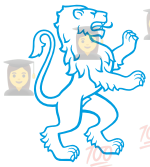


Kantonsschule Im Lee

Informatik

Skript

Cyril Wendl

Inhaltsverzeichnis

1 Zahlendarstellungen **1**

1.1 Natürliche Zahlen . . . 1

1.1.1 Ursprünge der Zahlendarstellung . . . 1

1.1.2 Basisgrössen . . . 2

1.1.3 Stellenwertdarstellung . . . 3

1.1.4 Umrechnungen zwischen Darstellungen . . . 7

1.1.5 Hexadezimalsystem . . . 7

1.1.6 Rechnen im Binärsystem . . . 8

2 Kodierungen **12**

2.1 Kodierung von Informationen in der Informatik . . . 12

2.1.1 Bits und Bytes . . . 12

2.1.2 Kodierung von Buchstaben: ASCII-Tabellen und UTF-8 . . . 13

2.1.3 Kodierung von Bildern . . . 16

2.1.4 Kodierung von Farben . . . 17

2.1.5 Kodierung von weiteren Dateiformaten . . . 20

2.1.6 Speicherplatz und Speicherbedarf . . . 20

Kapitel 1

Zahlendarstellungen

1.1 Natürliche Zahlen

1.1.1 Ursprünge der Zahlendarstellung

Die *natürlichen Zahlen* werden so genannt, da sie auf „natürliche Weise“ beim Zählen verwendet werden. Auf dem Gebiet der heutigen Demokratischen Republik Kongo wurde im 20. Jahrhundert ein Knochen, der sogenannte *Ishango-Knochen*, gefunden. Dieser Knochen wird auf die Zeit vor etwa 20'000 Jahren vor unserer Zeitrechnung datiert. In den Knochen wurden deutlich erkennbar von Menschen einige Kerben eingeritzt. Die Bedeutung dieser Kerben ist unklar, es wird jedoch angenommen, dass sie eine bestimmte Anzahl festhalten sollten.

Stellen wir uns vor, dass die Kerben die Anzahl der gefangenen Fische darstellen. Gefangene Fische durch Kerben in einem Knochen zu repräsentieren, verlangt bereits einen recht hohen Grad an Abstraktion — schliesslich haben gefangene Fische und Kerben in einem Knochen auf den ersten Blick keinen offensichtlichen Zusammenhang. Weniger abstrakt wäre es, eine Darstellung zu wählen, welche direkt an das zu zählende Objekt gebunden ist. In diesem Fall würde man also keine Kerben ritzen, sondern die entsprechende Anzahl Fische zeichnen / skizzieren. Eine Anzahl von drei Fischen könnte also durch drei Fisch-Symbole 🐟🐟🐟 festgehalten werden.

Anstelle von Kerben in einem Knochen oder Fisch-Symbole als Markierung könnte eine bestimmte Anzahl Fische auch durch die entsprechende Anzahl von anderen Markierungen wie Striche (|) oder Kieselsteine (•) repräsentiert werden:

$$\text{drei Kerben} \quad \leftrightarrow \quad 🐟🐟🐟 \quad \leftrightarrow \quad ||| \quad \leftrightarrow \quad \bullet\,\bullet\,\bullet$$

Die Darstellung durch skizzierte Fische ist am wenigsten abstrakt, da die dabei verwendete Markierung den zu zählenden Objekten (in diesem Fall also den Fischen) direkt nachempfunden ist. Zahlendarstellungen, welche nur ein Symbol / eine Markierung (Kerbe, 🐟, |, •) verwenden, werden ***unäre Zahlendarstellungen*** genannt.

Bemerkung 1.1:

(a) Der Vorteil von unären Darstellungen (gegenüber „komplexeren“ Darstellungen) ist ihre einfache Verständlichkeit und offensichtliche Interpretation.
(b) Unäre Darstellungen sind nur für kleine natürliche Zahlen übersichtlich, da ihre Darstellungslängen linear mit der Grösse der zu darstellenden Zahl wachsen (doppelt so grosse Zahl → doppelt so lange Darstellung).

(c) Unäre Darstellungen eignen sich nicht besonders gut zum Rechnen.

1.1.2 Basisgrössen

Anstelle von nur einem einzigen Symbol, wie bei unären Darstellungen, haben bereits Kulturen der Antike unterschiedliche ***Basisgrössen*** verwendet. „Basisgrössen“ bedeuten in diesem Kontext für Symbole, die gewisse Mengen, bzw. Zahlen repräsentieren, und dies unabhängig vom Objekt, das gezählt wird. Die Römer verwendeten die sieben verschiedenen Basisgrössen:

I (Eins), V (Fünf), X (Zehn), L (Fünfzig), C (Hundert), D (Fünfhundert) und M (Tausend).

Beispiel 1.1:
yexample:roman Die Zahl *fünfhundertvierundfünfzig* könnte in den römischen Basisgrössen zum Beispiel durch

$$CCCCCLIIII$$

oder durch

$$DLIIII$$

dargestellt werden.

Wir werden uns jede Basisgrösse als einen Typ von Münze von einem bestimmten Wert vorstellen. Die darzustellende Zahl denken wir uns dann als einen bestimmten Geldbetrag, den wir mithilfe dieser gegebenen Basisgrössen (Münztypen) auszahlen wollen. In dieser Vorstellung hatten die Römer also ein Währungssystem von sieben verschiedene Münztypen. Der Geldbetrag *einundzwanzig* könnte dann beispielsweise durch XXI (zwei Zehnermünzen und eine Einermünze) in antiker römischer Währung ausbezahlt werden. Die alten Ägypter hingegen verwendeten damals bereits Symbole (Zeichnungen) für die Basisgrössen $1, 10, 100, 1000, 10000, 100000$ und 1000000.

Ein Geldbetrag kann sowohl im römischen als auch ägyptischen System typischerweise auf mehrere unterschiedliche Arten ausbezahlt werden (zum Beispiel, indem nur Einer verwendet werden). Sie kennen diese Situation vom Bezahlen mit Bargeld — so kann der Betrag von hundert Schweizer Franken auf viele verschiedene Arten mit Schweizer Bargeld bezahlt werden (z.B. mit zwei Fünfzigernoten, fünf Zwanzigernoten, einer Fünfzigernote + zwei Zwanzigernoten + zwei Fünfliber, etc.). Natürlich ist es bei Zahlendarstellungen vorteilhaft, wenn sie *eindeutig* sind. Dies bedeutet, dass es für jede natürliche Zahl eine einzige minimale Münzenmenge gibt, die diese Zahl repräsentiert. Angenommen wir haben einen Betrag x gegeben, welcher durch unterschiedliche Ansammlungen von Münzen ausbezahlt werden kann. Dann betrachten wir diejenige Ansammlung von Münzen als die ***Zahlendarstellung*** von x, welche am wenigsten Münzstücke benötigt.

Nun gibt es aber denkbare Münzsysteme, in denen nicht jeder Betrag auf eindeutige Weise mit möglichst wenigen Münzstücken ausbezahlt werden kann. Dies werden Sie in Aufgabe 1.1 untersuchen.

Aufgabe 1.1

Erweitern Sie das römische Münzsystem um zwei weitere Münztypen Ihrer Wahl und finden Sie einen Betrag, der in Ihrem neuen System mehrere unterschiedliche Darstellungen mit derselben minimalen Anzahl von Münzen erlaubt.

Lösungsvorschlag zu Aufgabe 1.1

Wir erweitern das römische Münzsystem um die beiden Münztypen Ⅱ und Ⅲ. Dann besitzt der Betrag *vier* die beiden Darstellungen ⅠⅢ oder ⅡⅡ, welche beide eine (minimale) Anzahl von zwei Münzstücken verwenden.

Aufgabe 1.2

Ein Haupt-Nachteil von Zahlendarstellungen durch Münzsysteme ist, dass Münzsysteme für grosse Zahlen zur gleichen Ineffizienz führen wie unäre Zahlensysteme. Wie viele Zeichen würde man benötigen, um die Zahl 1'000'000'000 (1 Milliarde) im römischen Zahlensystem darzusetellen?

Lösungsvorschlag zu Aufgabe 1.2

Die Zahl wäre eine Million Stellen lang: Die Zahl M (Tausend) ist die grösste Zahl des römischen Zahlensystems und wir benötigen eine Million mal Tausend, um auf eine Milliarde zu kommen.

1.1.3 Stellenwertdarstellung

Das uns vermutlich am besten vertraute Zahlensystem ist das ***Dezimalsystem*** (Zehnersystem). Es verwendet die **zehn** verschiedenen Ziffern

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$$

und die (natürlichen) Potenzen $10^0, 10^1, 10^2, \ldots$ der Basis 10 als ***Stellenwerte***. Bereits in der Primarschule haben Sie gelernt, dass beispielsweise das Symbol 4605 definiert ist als die folgende Summe:

fünf Einer $(5 \cdot 10^0)$ + null Zehner $(0 \cdot 10^1)$ + sechs Hunderter $(6 \cdot 10^2)$ + vier Tausender $(4 \cdot 10^3)$.

Eine Ziffer gibt dabei jeweils an, wie viele Male die entsprechende Potenz der Basis 10 verwendet werden soll. Das Dezimalsystem hat also (die unendlich vielen verschiedenen) Basisgrössen

$$10^0 = 1, \quad 10^1 = 10, \quad 10^2 = 100, \quad 10^3 = 1000, \ldots$$

Definition 1.1:
ydefinition:Stellenwertdarstellung Seien $a_0, a_1, \ldots, a_n$ natürliche Zahlen. Dann ist der Ausdruck

$$a_n a_{n-1} \ldots a_1 a_0$$

definiert durch die Summe

$$a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0.$$

Beispiel 1.3:
Der Ausdruck 4205 entspricht der Zahl

$$4205 := \overbrace{4}^{a_3} \cdot \underbrace{10^3}_{=1000} + \overbrace{2}^{a_2} \cdot \underbrace{10^2}_{=100} + \overbrace{0}^{a_1} \cdot \underbrace{10^1}_{=10} + \overbrace{5}^{a_0} \cdot \underbrace{10^0}_{=1}.$$

Anstelle der Potenzen der Basis 10 kann man auch andere natürliche Zahlen als Basis verwenden. Das ***Binärsystem*** (Zweiersystem) verwendet zum Beispiel die Basis 2 und Ziffern aus $\{0, 1\}$.

Beispiel 1.4:
Wir wollen den Ausdruck 1101 in Basis 2 interpretieren. Um dies zu verdeutlichen, schreibt man die Basis tiefgestellt dahinter, also 1101_2. Der Ausdruck 1101_2 wird dann aufgefasst als die Zahl

$$1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 1 \cdot 8 + 1 \cdot 4 + 0 \cdot 2 + 1 \cdot 1 = 13.$$

Die binäre Zahl 1101_2 entspricht also der Zahl dreizehn.

Achtung

Wichtiger Hinweis 1.1:
Falls wir nicht explizit eine Basis tiefgestellt hinter einen Zahlenausdruck schreiben, dann ist dieser Ausdruck immer in Basis 10 (im Dezimalsystem) aufzufassen. Somit meinen wir also mit 11 die Zahl elf und nicht etwa die Zahl $11_2 = 1 \cdot 2^1 + 1 \cdot 2^0$.

Aufgabe 1.3

Schreiben Sie die Binärzahl 11011_2 als Dezimalzahl.

Lösungsvorschlag zu Aufgabe 1.3

$$11011_2 = 1 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 16 + 8 + 2 + 1 = 27$$

Aufgabe 1.4

Schreiben Sie die Dezimalzahl 14 als Binärzahl.

Lösungsvorschlag zu Aufgabe 1.4

$$14 = 8 + 4 + 2 = 1 \cdot 2^3 + 1 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 1110_2$$

Warum schränken wir uns beim Schreiben von Binärzahlen eigentlich auf den Gebrauch von lediglich den beiden Ziffern aus $\{0, 1\}$ ein? Ein Ausdruck der Form

$$3 \cdot 2^2 + 5 \cdot 2^1 + 7 \cdot 2^0$$

wäre doch ebenfalls sinnvoll. Diese Frage wollen wir in den folgenden Aufgaben klären. Erinnern Sie

sich zurück an Unterabschnitt 1.1.2. Dort haben wir versucht, Geldbeträge mit möglichst wenigen Münzstücken auszuzahlen.

Aufgabe 1.5

Der Ausdruck $3 \cdot 2^2 + 5 \cdot 2^1 + 7 \cdot 2^0$ verwendet $3+5+7 = 15$ Münzstücke und stellt die Zahl 29 dar. Stellen Sie die Zahl 29 im Binärsystem dar, verwenden Sie diesmal aber lediglich Ziffern aus $\{0, 1\}$. Wie viele Münzstücke verwendet diese Darstellung?

Lösungsvorschlag zu Aufgabe 1.5

$$29 = 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0$$

Diese Darstellung verwendet lediglich 5 Münzstücke.

Beispiel 1.5:
yexample:ersetzen Wir wollen dreizehn Objekte im Dezimalsystem mit so wenigen Münzstücken wie möglich darstellen. Wir könnten dies mit dreizehn Einern tun. Doch wir sehen sofort, dass sich zehn Einer stets durch nur einen einzigen Zehner (und null Einer) ersetzen lassen. Dadurch haben wir insgesamt neun Münzen gespart:

- verfügbare Münztypen: ①, ⑩, ...
- Münzen vorher:
  ①①①①①
  ①①①①①
  ①①①
- Münzen nachher:
  ⑩①①①

Definition 1.2 (b-adische Zahlendarsellung für natürliche Zahlen)**:**
ydefinition:badisch Gegeben ist eine natürliche Zahl $b \geq 2$ (***Basis***). Dann definieren wir

$$(a_n a_{n-1} \dots a_1 a_0)_b := a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \dots + a_1 \cdot b^1 + a_0 \cdot b^0,$$

wobei die Zahlen $a_0, a_1, \dots, a_{n-1}, a_n \in \{0, 1, \dots, b-1\}$ ***Ziffern*** genannt werden. Wird nicht explizit eine andere Basis genannt, so ist stets die Basis $b = 10$ gemeint.
Diese Zahlendarstellung wird b-***adische Zahlendarstellung*** genannt.

Beispiel 1.6:
Für die Wahl $b := 3$ erhalten wir die 3-adische Zahlendarstellung (Dreiersystem). Die Zahl 211_3 entspricht dann der Zahl

$$211_3 = 2 \cdot 3^2 + 1 \cdot 3^1 + 1 \cdot 3^0 = 2 \cdot 9 + 1 \cdot 3 + 1 \cdot 1 = 22.$$

Aufgabe (Challenge) 1.6

Begründen Sie, warum es nicht besonders sinnvoll ist, die natürliche Zahl 1 als Basis zu wählen, selbst wenn wir in **??** die Einschränkung $a_0, a_1, \ldots, a_{n-1}, a_n \in \{0, 1, \ldots, b-1\}$ an die Ziffern ignorieren?

Lösungsvorschlag zu Aufgabe 1.6

Mit der Wahl der Basis $b := 1$ erhalten wir keine unterschiedlichen Stellenwerte, da $b^k = 1^k = 1$ für alle natürlichen Zahlen k gilt.

Bemerkung 1.2:
Warum ist es sinnvoll in **??** zu fordern, dass eine b-adische Darstellung keine Ziffern grösser als $b-1$ verwenden darf? Unsere Argumentation ist ähnlich zu der in **??**. Angenommen eine Darstellung

$$a_n \cdot b^n + a_{n-1} \cdot b^{n-1} + \ldots + a_1 \cdot b^1 + a_0 \cdot b^0$$

würde eine Ziffer $a_k \geq b$ verwenden. Anders gesagt, würden dann b oder mehr Münzstücke von dem Typ b^k verwendet werden. Dann könnten wir aber immer eine Gruppe von b vielen Münzen vom Typ b^k durch jeweils eine einzige Münze vom Typ b^{k+1} ersetzen und würden für jede solche Gruppe $b-1$ Münzstücke einsparen.
Beispielsweise ist es in der Basis $b = 10$ nicht sinnvoll 23 Hunderter $(23 \cdot 10^2)$ zu verwenden, da wir stattdessen zwei Tausender und nur drei Hunderter $(2 \cdot 10^3 + 3 \cdot 10^2)$ verwenden könnten. Dabei würden wir insgesamt 18 Münzstücke einsparen.

Aufgabe 1.7

Ordnen Sie die folgenden Zahlen aufsteigend:

$$43_{10}, 2201_3, 11101_2, 33_4, 142_5, 111_2.$$

Lösungsvorschlag zu Aufgabe 1.7

$$\begin{aligned}
2201_3 &= 2 \cdot 3^3 + 2 \cdot 3^2 + 0 \cdot 3^1 + 1 \cdot 3^0 = 73, \\
11101_2 &= 1 \cdot 2^4 + 1 \cdot 2^3 + 1 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 29, \\
33_4 &= 3 \cdot 4^1 + 3 \cdot 4^0 = 15, \\
142_5 &= 1 \cdot 5^2 + 4 \cdot 5^1 + 2 \cdot 5^0 = 47, \\
111_2 &= 1 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0 = 7, \\
&\Rightarrow 111_2 < 33_3 < 11101_2 < 43 < 142_5 < 2201_3
\end{aligned}$$

Aufgabe (Challenge) 1.8

Die nachfolgende Aussage gilt für jede Basis b eines b-adischen Systems. Zur Vereinfachung werden wir aber nur die Wahl der Basis $b := 10$ untersuchen. Zeigen Sie, dass die 10-adische Darstellung eindeutig ist. Es gibt also keine zwei unterschiedlichen 10-adischen Darstellungen x und y, welche dieselbe Zahl (Anzahl Objekte) darstellen.

Lösungsvorschlag zu Aufgabe 1.8

Seien

$$x := a_n \cdot 10^n + a_{n-1} \cdot 10^{n-1} + \ldots + a_k \cdot 10^k + a_{k-1} \cdot 10^{k-1} + \ldots + a_1 \cdot 10^1 + a_0 \cdot 10^0$$
$$y := b_n \cdot 10^n + b_{n-1} \cdot 10^{n-1} + \ldots + b_k \cdot 10^k + b_{k-1} \cdot 10^{k-1} + \ldots + b_1 \cdot 10^1 + b_0 \cdot 10^0$$

zwei unterschiedliche 10-adische Darstellungen derselben Zahl. Dann müssen sich x und y an mindestens einer Stelle (in einer Ziffer) unterscheiden. Sei 10^k die grösste Stelle, an der sich x und y unterscheiden. Wir können annehmen, dass $a_k > b_k$ gilt. Dann bewirkt der Unterschied an dieser Stelle eine Differenz von mindestens 10^k. Doch selbst mit der Wahl $a_{k-1} = \ldots = a_1 = a_0 = 0$ und $b_{k-1} = \ldots = b_1 = b_0 = 9$ kann nur noch ein Unterschied von $10^k - 1$ erwirkt werden. Dann ist aber die von x dargestellte Zahl um mindestens 1 grösser als die von y dargestellte Zahl.

1.1.4 Umrechnungen zwischen Darstellungen

Wir wollen nun sehen, wie verschiedene Darstellungen ineinander umgerechnet werden können.

Beispiel 1.7 (Darstellung in Basis 5)**:**
yexample:umrechnung Wie lautet die 5-adische Darstellung der Zahl 84? Wir beginnen damit alle natürlichen Potenzen von 5 aufzulisten, die kleiner oder gleich 84 sind:

$$5^0 = 1, \quad 5^1 = 5, \quad 5^2 = 25.$$

Die Potenz $5^3 = 125 > 84$ ist bereits grösser als die gegebene Zahl. Nun versuchen wir möglichst viele der möglichst grössten Potenz von 5 zu verwenden, solange diese noch in 84 passen. Die grösste mögliche Potenz ist $5^2 = 25$ und 25 hat dreimal in 84 Platz. Es bleibt also noch $84 - 3 \cdot 5^2 = 9$ übrig. Wir verwenden möglichst viele der möglichst grössten Potenz von 5, die in 9 Platz haben und stellen fest, dass $5^1 = 5$ einmal Platz hat. Dann bleibt noch $9 - 5^1 = 4$ übrig. Wir verwenden möglichst viele der möglichst grössten Potenz von 5, die in 4 Platz haben. Wir verwenden also viermal die Potenz $5^0 = 1$. Nun bleibt nichts mehr übrig und die 5-adische Darstellung von 84_{10} lautet:

$$84_{10} = 314_5.$$

Da wir in **??** immer mit der grössten Basisgrösse begonnen haben und so viele Münzen wie möglich davon verwendet haben, nennen wir diese Methode ***gierig*** (englisch: ***greedy***).

1.1.5 Hexadezimalsystem

Wie Sie in Unterabschnitt 1.1.3 gesehen haben, kann jede Zahl im Grunde genommen mit (fast) beliebig vielen Ziffern dargestellt werden. Während das Binärsystem die Ziffern $\{0, 1\}$ verwendet,

verwenden wir im (uns geläufigeren) Dezimalsystem die Ziffern $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. Wir haben zudem auch beobachtet, dass die höchste Ziffer in einem Zahlensystem mit b Ziffern immer $b - 1$ ist: Beispielsweise ist im Binärsystem, das 2 Ziffern hat, die höchste Ziffer eine 1 $(= 2 - 1)$. Im Dezimalsystem, das bekanntlich 10 Ziffern hat, ist die höchste Ziffer eine 9 $(= 10 - 1)$. Die höchste Ziffer in einem b-adischen System ist also immer $b - 1$.

Welche Ziffern sollen wir jedoch wählen, sofern $b > 10$ gewählt wird? Die dezimale Zahl 10 ist ja keine Ziffer, sondern besteht selber aus zwei Ziffern (genau so, wie es das Dezimalsystem vorschreibt).

Im Grunde genommen sind Ziffern wie „1“, „2“ usw. nichts weiter als „Zeichnungen“, die von Menschen angefertigt worden sind, und mit welchen eine gewisse Menge, bzw. Anzahl gemeint ist. Wir könnten also einfach andere, uns bekannte „Zeichnungen“ verwenden, um höhere Mengen zu bezeichnen, also etwa Buchstaben. Genau dies wird im sogenannten ***Hexadezimalsystem*** gemacht $(b = 16)$. Die möglichen Zeichen im Hexadezimalsystem sind:

$$\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F\},$$

wobei beispielsweise die Ziffer B die (dezimale) Zahl 11 (Elf) bezeichnet.

Zur Umrechnung von Zahlen vom einen in das andere System verfolgen wir denselben Ansatz wie in **??**

Aufgabe 1.9

Wie lautet die Zahl 27_{10} in der 16-adischen Darstellung?

Lösungsvorschlag zu Aufgabe 1.9

$$27_{10} = 1 \cdot 16^1 + 11 \cdot 16^0 = 1B_{16}$$

Aufgabe 1.10

Geben Sie folgende Zahlen im hexadezimalen System aus: 44, 23, 90, 124, 306.

Lösungsvorschlag zu Aufgabe 1.10

2C, 17, 5A, 7C, 132

1.1.6 Rechnen im Binärsystem

Rechnen im Binärsystem funktioniert genauso, wie Sie es für das Dezimalsystem gelernt haben. Beispielsweise, wenn Sie die Zahlen 49035_{10} und 718293_{10} addieren:

$$\begin{array}{rcccccc} & 0 & 4 & 9 & 0 & 3 & 5_{10} \\ + & 7 & \overset{1}{1} & 8 & \overset{1}{2} & 9 & 3_{10} \\ \hline & 7 & 6 & 7 & 3 & 2 & 8_{10} \end{array}$$

Die kleinen Ziffern in der Rechnung oben stellen den Übertrag (Englisch: *carry over* oder einfach nur *carry*) dar. Ähnlich dazu gehen Sie im Binärsystem vor. Dabei müssen im Wesentlichen nur folgende 5 Regeln beachtet werden:

- $0 + 0 = 0$
- $0 + 1 = 1$
- $1 + 0 = 1$
- $1 + 1 = 0$, Übertrag 1
- $1 + 1 + 1 = 1$, Übertrag 1

Ein Beispiel sei unten angegeben:

$$\begin{array}{cccccccc} & 1 & 0 & 1 & 0 & 1 & 1_2 \\ + & \overset{1}{1} & \overset{1}{1} & \overset{0}{1} & \overset{1}{0} & \overset{0}{1} & 0_2 \\ \hline 1 & 1 & 0 & 0 & 1 & 0 & 1_2 \end{array}$$

Aufgabe 1.11

Rechnen Sie folgende beiden Zahlen im Binärsystem zusammen: 10001010_2, 11101_2.

Lösungsvorschlag zu Aufgabe 1.11

10100111_2

Aufgabe 1.12

Welches ist die grösste 4-stellige Binärzahl (in Dezimalschreibweise)?

Lösungsvorschlag zu Aufgabe 1.12

$$1111_2 = 2^4 + 2^3 + 2^2 + 2^1 = 8 + 4 + 2 + 1 = 15_{10}$$

Aufgabe (Challenge) 1.13

Es sei $n > 0$ eine natürliche Zahl. Es sei z die grösste n-stellige Binärzahl. Wie sieht z aus? Finden Sie eine möglichst einfache Darstellung von z.

✔ Lösungsvorschlag zu Aufgabe 1.13

Die grösste n-stellige Binärzahl z ist gegeben durch

$$z := \underbrace{1\ldots11_2}_{n\text{-Stellen}} = 1\cdot 2^0 + 1\cdot 2^1 + 1\cdot 2^2 + \ldots + 1\cdot 2^{n-1}.$$

Die Umrechnung in Basis 10 ist aufwendig, da wir die Summe $2^0 + 2^1 + 2^2 + \ldots + 2^{n-1}$ berechnen müssten. Lassen Sie uns die Zahl $z + 1$ bestimmen:

$$\begin{array}{cccccccc} & 1 & 1 & \ldots & 1 & 1 & 1_2 \\ + & 0 & 0 & \ldots & 0 & 0 & 1_2 \\ & {\scriptstyle 1} & {\scriptstyle 1} & & {\scriptstyle 1} & {\scriptstyle 1} & \\ \hline 1 & 0 & 0 & \ldots & 0 & 0 & 0_2 \end{array}$$

Wir haben die kleinste $n+1$-stellige Binärzahl erhalten. Wir sehen nun, dass $z+1$ geben ist durch

$$z + 1 = \underbrace{10\ldots00_2}_{n+1\text{-Stellen}} = 0\cdot 2^0 + 0\cdot 2^1 + \ldots + 0\cdot 2^{n-1} + 1\cdot 2^n = 2^n.$$

Es fallen also alle Summanden bis auf den Term $1\cdot 2^n$ weg. Da z genau 1 kleiner ist als $z+1$, finden wir $z = 2^n - 1$.

Aufgabe 1.14

Rechnen Sie folgende beiden Zahlen im Binärsystem zusammen: 11110001_2, 00001111_2.

✔ Lösungsvorschlag zu Aufgabe 1.14

Die mathematisch richtige Antwort wäre eigentlich 100000000_2. Falls wir die Rechnung jedoch in einer 8-Bit-Umgebung ausführen (wie es die ersten Computer waren), könnte das Resultat auch 0 sein, da die Zahl zu lang würde (Überlauf, en. *overflow*).

Aufgabe (Challenge) 1.15

- Wie lange benötigen Sie für 6 Umwandlungen? Testen Sie sich hier.
- Testen Sie Ihre Zahlensystem-Umwandlungs-Wissen hier!
- Schauen Sie sich das Video der Die Marble Adding Machine an.
- Schauen Sie sich an, wie man einen binären Computer auch mit Wasser bauen kann.

Kapitel 2

Kodierungen

2.1 Kodierung von Informationen in der Informatik

Wir haben in den vorausgegangenen Kapiteln gelernt, wie man Zahlen im Binärsystem sowie in anderen b-adischen Systemen darstellt. Wir möchten jedoch selbstverständlich nicht nur Zahlen, sondern auch andere digitale Informationen wie etwa Buchstaben oder Bilder darstellen können. Im folgenden Kapitel widmen wir uns daher der Darstellung von Buchstaben und Bildern, sowie weiteren Dateiformaten im Allgemeinen.

2.1.1 Bits und Bytes

In Computern treten Bits oft in 8er-Gruppen auf. Einen solchen Block von 8 Bits nennt man ein Byte. Der Ausdruck stammt aus dem Englischen und setzt sich aus einem Wortspiel von *bit* (= *binary digit*), „binäre Ziffer“, und *bite* = „Bissen“ zusammen (die Schreibweise „byte“ wurde gewählt, um eine Verwechslung mit dem englischen Wort *bite* zu vermeiden).

Folgende Zahl ist die binäre Repräsentation der Dezimalzahl 22:

$$00010110_2 = 22_{10}$$

Bei Dezimalzahlen sind wir daran gewohnt, führende Nullen einfach wegzulassen (man könnte schliesslich genauso gut 000022 schreiben) – bei Binärzahlen werden üblicherweise alle Bits ausgeschrieben (also 00010110, nicht 10110).

Aufgabe 2.1

Wie wird die Zahl 42_{10} mit einem Byte dargestellt? Vergessen Sie die führenden Nullen nicht!

Lösungsvorschlag zu Aufgabe 2.1

00101010_2

Aufgabe 2.2

Welches ist die grösste Dezimalzahl, die mit einem Byte (= 8 Bits) ausgedrückt werden kann?

Lösungsvorschlag zu Aufgabe 2.2

$255_{10} = 11111111_2$

2.1.2 Kodierung von Buchstaben: ASCII-Tabellen und UTF-8

In den vorausgehenden Kapiteln haben wir gelernt, dass Zahlen in unterschiedlichen Zahlensystemen unterschiedlich repräsentiert werden können. Insbesondere haben wir uns das Binärsystem genauer angeschaut, da elektronische Rechner typischerweise Informationen in binärer Form verarbeiten.

Wenn Computer lediglich Nullen und Einsen abspeichern können, wie kann man damit Zahlen und Buchstaben repräsentieren? Genau dies wurde umgesetzt mit einer der ersten ***Text-Kodierungen***, dem American Standard Code for Information Interchange (ASCII).

Die Grundidee der ASCII-Kodierung war, ein Byte pro Zeichen zu verwenden. Ein Bit war allerdings reserviert für weitere Zwecke, wie beispielsweise die Fehlerkorrektur (Paritätsbit). Daher stehen in der originalen ASCII-Kodierung lediglich 7 Bits zur Kodierung von Buchstaben zu Verfügung.

Aufgabe 2.3

Wie viele unterschiedliche Zeichen kann man mit der ASCII-Kodierung abbilden, wenn man nur 7 von 8 Bits zur Kodierung nutzt?

Lösungsvorschlag zu Aufgabe 2.3

128 Zeichen, da $2^7 = 128$.

Aufgabe 2.4

Welches ist die grösste binäre Zahl, die man mit 8 Bits (einem Byte) darstellen kann? Wie lautet diese Zahl in hexadezimaler Darstellung?

Lösungsvorschlag zu Aufgabe 2.4

Die grösste binäre Zahl mit 8 Bit ist $11111111_2 = 255_{10} = FF_{16}$. Wir sehen also, dass man mit lediglich zwei Ziffern alle binären Codes eines Bytes darstellen kann. Unter anderem wird auch aus diesem Grund häufig die hexadezimale Darstellung für Byte-Codes verwendet.

Leider reichen die wenige Zeichen, welche mit dem ASCII-Standard kodiert werden können, bei weitem nicht aus, um die Vielfalt an heutigen Zeichen (in unterschiedlichen Sprachen), Spezialzeichen und Emojis darzustellen – daher wurde ASCII mittlerweile auf breiter Front von einem neuen Standard abgelöst – Unicode Transformation Format — 8-bit (UTF-8). Bei diesem Standard stehen für jedes Zeichen bis zu 4 Bytes zur Verfügung.

Aufgabe 2.5

Wie viele unterschiedliche Zeichen kann man mit vier Bytes maximal abbilden?

Lösungsvorschlag zu Aufgabe 2.5

2^{32} =4'294'967'296 mögliche Zeichen. Allerdings kann UTF-8 etwas weniger Zeichen abbilden, da im ersten Byte noch Speicherplatz eingerechnet werden muss zur Angabe, ob das Zeichen mit 1, 2, 3 oder 4 Bytes angegeben wird.

Dezimal	Hexadezimal	Binär	Zeichen
0	00	00000000	NUL
1	01	00000001	SOH
2	02	00000010	STX
3	03	00000011	ETX
4	04	00000100	EOT
5	05	00000101	ENQ
6	06	00000110	ACK
7	07	00000111	BEL
8	08	00001000	BS
9	09	00001001	HT
10	0A	00001010	LF
11	0B	00001011	VT
12	0C	00001100	FF
13	0D	00001101	CR
14	0E	00001110	SO
15	0F	00001111	SI
16	10	00010000	DLE
17	11	00010001	DC1
18	12	00010010	DC2
19	13	00010011	DC3
20	14	00010100	DC4
21	15	00010101	NAK
22	16	00010110	SYN
23	17	00010111	ETB
24	18	00011000	CAN
25	19	00011001	EM
26	1A	00011010	SUB
27	1B	00011011	ESC
28	1C	00011100	FS
29	1D	00011101	GS
30	1E	00011110	RS
31	1F	00011111	US
32	20	00100000	Space
33	21	00100001	!
34	22	00100010	”
35	23	00100011	#
36	24	00100100	$
37	25	00100101	%
38	26	00100110	&
39	27	00100111	’
40	28	00101000	(
41	29	00101001	)
42	2A	00101010	*
43	2B	00101011	+
44	2C	00101100	,
45	2D	00101101	-
46	2E	00101110	.
47	2F	00101111	/
48	30	00110000	0
49	31	00110001	1
50	32	00110010	2
51	33	00110011	3
52	34	00110100	4
53	35	00110101	5
54	36	00110110	6
55	37	00110111	7
56	38	00111000	8
57	39	00111001	9
58	3A	00111010	:
59	3B	00111011	;
60	3C	00111100	<
61	3D	00111101	=
62	3E	00111110	>
63	3F	00111111	?

Dezimal	Hexadezimal	Binär	Zeichen
64	40	01000000	@
65	41	01000001	A
66	42	01000010	B
67	43	01000011	C
68	44	01000100	D
69	45	01000101	E
70	46	01000110	F
71	47	01000111	G
72	48	01001000	H
73	49	01001001	I
74	4A	01001010	J
75	4B	01001011	K
76	4C	01001100	L
77	4D	01001101	M
78	4E	01001110	N
79	4F	01001111	O
80	50	01010000	P
81	51	01010001	Q
82	52	01010010	R
83	53	01010011	S
84	54	01010100	T
85	55	01010101	U
86	56	01010110	V
87	57	01010111	W
88	58	01011000	X
89	59	01011001	Y
90	5A	01011010	Z
91	5B	01011011	[
92	5C	01011100	\
93	5D	01011101	]
94	5E	01011110	^
95	5F	01011111	_
96	60	01100000	‘
97	61	01100001	a
98	62	01100010	b
99	63	01100011	c
100	64	01100100	d
101	65	01100101	e
102	66	01100110	f
103	67	01100111	g
104	68	01101000	h
105	69	01101001	i
106	6A	01101010	j
107	6B	01101011	k
108	6C	01101100	l
109	6D	01101101	m
110	6E	01101110	n
111	6F	01101111	o
112	70	01110000	p
113	71	01110001	q
114	72	01110010	r
115	73	01110011	s
116	74	01110100	t
117	75	01110101	u
118	76	01110110	v
119	77	01110111	w
120	78	01111000	x
121	79	01111001	y
122	7A	01111010	z
123	7B	01111011	{
124	7C	01111100	|
125	7D	01111101	}
126	7E	01111110	~
127	7F	01111111	DEL

Tabelle 2.1: Darstellbare Zeichen der originalen ASCII-Kodierung able:ascii

Aufgabe 2.6

Im Computer ist die folgende Bit-Folge gespeichert:

`01101000 01100101 01101100 01101100 01101111`

Nutzen Sie **??**, um die Bit-Folge in Zeichen zu übersetzen. Welches Wort wird damit codiert?

Tipp: Nutzen Sie die Suchfunktion (ctrl + F, oder ⌘ + F) um in der ASCII-Tabelle nach den Bitfolgen zu suchen.

Lösungsvorschlag zu Aufgabe 2.6

hello

Aufgabe 2.7

Schreiben Sie Ihren Namen in der ASCII-Kodierung.

2.1.3 Kodierung von Bildern

Auch Bilder werden im Computer durch Nullen und Einsen repräsentiert, üblicherweise indem die einzelnen Bildpunkte (= Pixel) in eine bestimmte Ordnung gebracht werden (meist Durchzählen von links nach rechts, dann von oben nach unten), und für jedes Pixel wird dann einfach ein Farbwert gespeichert – also eine (Binär-)Zahl die angibt, welche Farbe dieses Pixel hat.

Wenn es sich um ein Schwarz-Weiss-Bild handelt ist dies relativ einfach, wie **??** zeigt.

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

Tabelle 2.2: Beispiel eines Schwarz-Weiss-Bilds
able:bw-img

Aufgabe 2.8

Um was für ein Bild handelt es sich bei **??**?

Lösungsvorschlag zu Aufgabe 2.8

Um ein Kreuz.

Aufgabe 2.9

Wie viele Bit Speicherplatz benötigt das Bild?

Lösungsvorschlag zu Aufgabe 2.9

25 ($= 5 \cdot 5$).

Aufgabe 2.10

Angenommen, für jeden Pixel stünden nicht nur Schwarz und Weiss, sondern insgesamt 8 Graustufen zur Verfügung. Wie viel Speicherplatz (in Bits) wäre dann nötig, um das Bild zu speichern?

Lösungsvorschlag zu Aufgabe 2.10

75, da 3 Bits ausreichen, um 8 verschiedene Stufen abzubilden (von 0 bis 7, $7_{10} = 111_2$). Daher haben wir und 3 Bits * 25 Pixel = 75 Bits.

Aufgabe 2.11

Graphics Interchange Format (GIF)-Bilder im Internet können in der Regel 256 Farben unterscheiden. Wie viele Bits pro Pixel sind dafür nötig?

Lösungsvorschlag zu Aufgabe 2.11

8 (siehe Lösung auf Aufgabe 2.2).

2.1.4 Kodierung von Farben

Computerbildschirme bestehen aus winzigen Lampen (typischerweise in Light-Emitting Diode (LED)-Technologie), die entweder rot, grün oder blau leuchten können. Falls Sie zu Hause einen alten Bildschirm haben, können Sie den gerne von nahe betrachten und sollten die einzelnen LEDs erkennen können (siehe **??**).

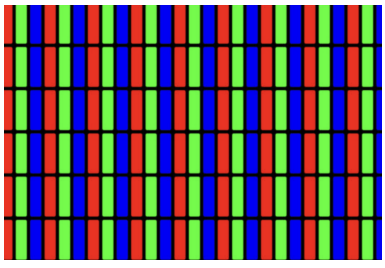
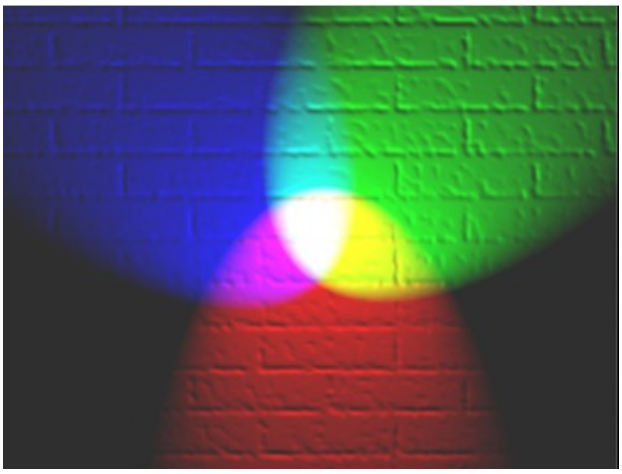
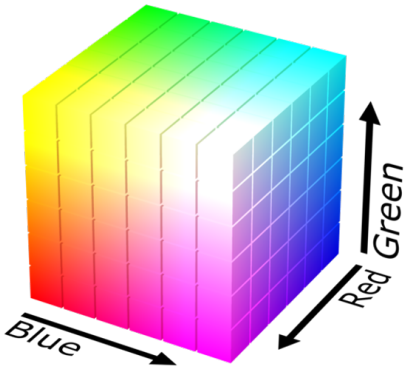


Abbildung 2.1: LED-Leuchtmittel in einem Computer-Bildschirm
ig:pixels

Durch die Mischung der jeweiligen Leucht-Intensitäten kann eine Vielzahl möglicher Farben dargestellt werden (siehe **??**).



(a) Lichtkegel machen dasselbe wie ein Monitor ig:rgb-farbmischung-1ub@fig:rgb-farbmischung-1



(b) Mögliche Farbmischungen ig:rgb-farbmischung-2ub@fig:rgb-farbmischung-2

Abbildung 2.2: Prinzip der additiven Farbmischung in einem Computer-Monitor ig:rgb-farbmischung

Wie weiss nun ein Bildschirm, welche Pixels wie hell leuchten sollen? Auch hier gilt: Da Computer nur mit Nullen und Einsen arbeiten können, wird jedes Pixel mit einer Zahl angesteuert, die die Helligkeit in % bezeichnet. Moderne Bildschirme können eine Vielzahl von Farben abbilden.

Beispiel 2.1:
Da jedes Farb-Pixel eines Bildschirms typischerweise in 256 Helligkeitsstufen leuchten kann, braucht es 8 Bit (= ein Byte) pro Farbe, um die Helligkeitsstufe zu kodieren, da $2^8 = 256$. Da pro „gemischte“ Farbe 3 Pixel leuchten, braucht es also $3 \cdot 8$ Bits, d.h. 24 Bits, um eine Farbe zu kodieren. Daraus ergibt sich, dass ein Monitor auf jedem Pixel $2^{24} \approx 16.7$ Millionen Farben darstellen kann.

Beispiel 2.2:
Da die Darstellung einer Farbe im Binärsystem 24 Stellen braucht, braucht es etwas Zeit, um diese Darstellung zu lesen. Stattdessen wird häufig die hexadezimale Schreibweise verwendet:
`11111111000000000000000` = `#FF0000` = `255, 0, 0` = reines Rot
`100000001000000010000000` = `#808080` = `128, 128, 128` = grau
Mit dem Hashtag (`#`) vor der Farbe wird klargestellt, dass die Farbwerte als hexadezimale Zahlen dargestellt sind.

Aufgabe 2.12

Ordnen Sie die folgenden hexadezimalen Farbkodierungen den Farben zu:

- `#FF0000`
- `#00FF00`
- `#0000FF`
- `#000000`
- `#FFFFFF`

- Schwarz
- Grün
- Blau
- Rot
- Weiss

Aufgabe (Challenge) 2.13

Erraten Sie weitere Farben unter diesem Link.

Jedes Pixel besteht also aus 3 Farben, die jeweils mit einem Byte kodiert werden. Die Anzahl Bits pro Pixel bestimmt die ***Farbtiefe*** eines Bilds (siehe **??**).

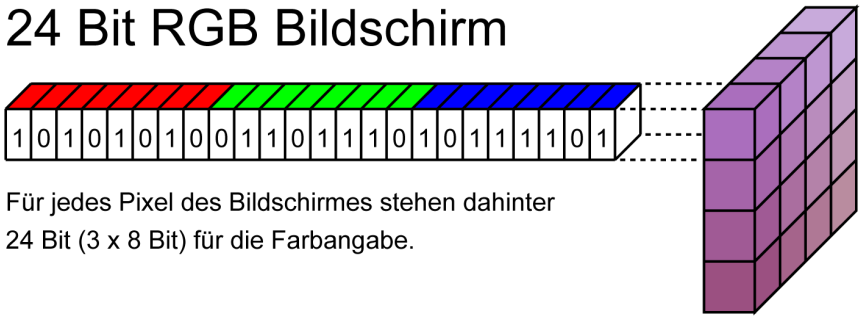
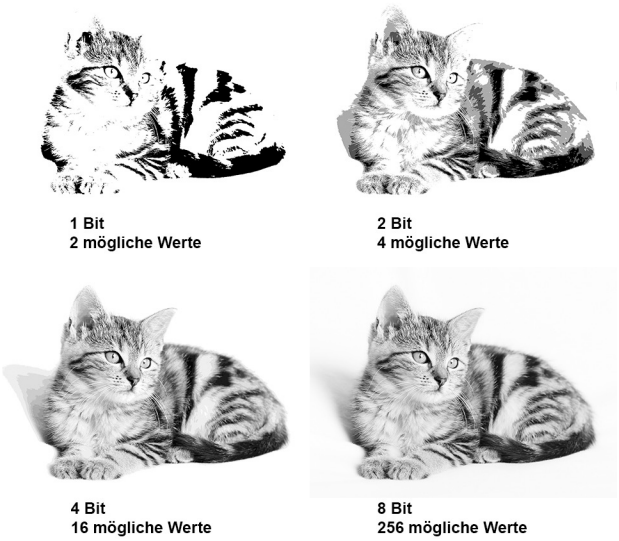


Abbildung 2.3: Farbtiefe in einem Bild
ig:farbtiefe

Je geringer die Farbtiefe, desto weniger fein können Farben oder Grautöne abgestuft werden (siehe **??**).

(a) Farbtiefe: Skala
ig:colordepth-barub@fig:colordepth-bar



(b) Farbtiefen bei Graustufen-Bildern
ig:colordepth-catub@fig:colordepth-cat

Abbildung 2.4: Farbtiefe: Beispiele
ig:colordepth-examples

Gewisse Bildformate sparen durch Indexierung und Kompression Speicherplatz. Falls beispielsweise ein grosser Teil des Bilds aus blauem Himmel besteht, muss die Farbe nicht für jedes einzelne Pixel abgespeichert werden. Mit dem Thema Kompression vertiefen wir uns in einem späteren Teil.

Aufgabe 2.14

Wie viel Speicherplatz braucht ein $500 \cdot 500$ Pixel grosses Bild, wenn nur 64 Farben unterschieden werden sollen?

Lösungsvorschlag zu Aufgabe 2.14

Das Bild hat 250'000 Pixel, und jedes Pixel braucht 6 Bit (da $2^6 = 64$). Das ergibt insgesamt $250'000 \cdot 6 = 1'500'000$ Bits, oder umgerechnet 187'500 Bytes (= 187.5 KB).

2.1.5 Kodierung von weiteren Dateiformaten

Wir wissen nun, das Computer nur mit Nulles und Einsen „arbeiten". Woher wissen wir aber, ob eine Folge aus Nullen und Einsen ein Bild, eine Zahl, ein Buchstabe oder etwas anderes repräsentiert? Hier kommen Dateiformate ins Spiel (siehe **??**).

In der Datei	**Bedeutung (z.B.)**
`dateiname.txt`	die Zeichen: < | . . .
`dateiname.doc`	Teil des Befehls: „Einzug links um 1.75 cm"
`dateiname.xls`	die Zahl 9662249 in einer Zelle
`dateiname.gif`	3 Bildpunkte: blau, hellblau, weinrot (`.gif` kennt nur 256 Farben → 1 Byte pro Pixel)
`dateiname.bmp`	1 Bildpunkt: rötlich-braun (16.7 Mio. Farben → 3 Bytes pro Pixel)

Tabelle 2.3: Dateiformate und Kodierungen: Bedeutung einer Bitsequenz (z.B. `10010011 01101111 00101001`)

able:formats-codes

Aufgabe 2.15

Um zu sehen, dass all Ihre Dateien tatsächlich nur aus Nullen oder Einsen bestehen, können Sie sich dieses Video anschauen.

2.1.6 Speicherplatz und Speicherbedarf

Wir haben bereits gelernt, dass ein Byte acht Bits entspricht. Heutzutage rechnen wir jedoch mit höheren Grössenordnungen (siehe **??**).

Masseinheit	**Dezimalsystem**		Grössenordnung
KB (Kilobyte)	10^3 B(yte)	1'000 B	eine Text-Datei
MB (Megabyte)	10^6 B	1'000'000 B	eine Musik-Datei
GB (Gigabyte)	10^9 B	1'000'000'000 B	eine Video-Datei
TB (Terabyte)	10^{12} B	1'000'000'000'000 B	kleiner Firmen-Server
PB (Petabyte)	10^{15} B	...	Facebook-Server (Meta)
EB (Exabyte)	10^{18} B	...	alle CERN-Daten
ZB (Zettabyte)	10^{21} B	...	alle Daten ($\sim$ 100 ZB)
YB (Yottabyte)	10^{24} B	...	$\approx$ 2030?

Tabelle 2.4: Gängige Daten-Grössenordnungen in der Informatik (ein **B**yte = 8 **b**it)
able:daten-groessenordnungen

Aufgabe 2.16

Welches ist der übliche / gebräuchliche Name für ein „Megagramm"?

Lösungsvorschlag zu Aufgabe 2.16

eine Tonne

Aufgabe (Challenge) 2.17

Weshalb haben Festplatten häufig eine leicht kleinere Speichergrösse als angegeben, insbesondere unter Windows? Lesen Sie dazu diesen Support-Artikel von Apple.

Lernziele: Zahlensysteme und Kodierungen

- ☐ Ich kann Zahlen von einem gegebenen Zahlensystem in das Dezimalsystem umwandeln. Beispiele
  - $101_2 = 5_{10}$
  - $204_5 = 54_{10}$
  - $1A_{16} = 26_{10}$
- ☐ Ich kann Zahlen vom Dezimalsystem in ein vorgegebenes Zahlensystem umwandeln. Beispiele:
  - $5_{10} = 101_2$
  - $135_{10} = 1020_5$
  - $26_{10} = 1A_{16}$
- ☐ Ich kann Zahlen in einem fremden Zahlensystem (schriftlich) addieren. Zum Beispiel:

$$123_5 + 431_5 = 1104_5$$

- ☐ Ich kenne die Einheiten Bits und Bytes und kann Beispiele nennen, wie diese physikalisch realisiert werden können.
- ☐ Ich kann berechnen, wie viele unterschiedlichen Informationen mit einer gewissen Anzahl Bits codiert werden können.
- ☐ Ich kann erklären, worum es sich bei ASCII handelt (Sie müssen die Tabelle aber nicht auswendig kennen 😉).
- ☐ Ich verstehe Bilder als ein Raster aus Pixeln, die je eine bestimmte Farbe haben.
- ☐ Ich kenne den Begriff der Farbtiefe und kann berechnen, wie viele Farben pro Pixel bei einer gegebenen Farbtiefe möglich sind.
- ☐ Ich kann berechnen, wie viele Bits nötig sind, um ein Bild mit einer bestimmten Anzahl möglichen Farben zu speichern.
- ☐ Ich kann erklären, wie man eine RGB-Farbcodierung wie `#1243A3` interpretieren kann.
- ☐ Ich kann einen hexadezimalen Farbcode wie `#1243A3` einer Farbe (aus einer Auswahl) zuordnen.

Glossar

ASCII American Standard Code for Information Interchange. 13, 15, 16

GIF Graphics Interchange Format. 17

LED Light-Emitting Diode. 17

UTF-8 Unicode Transformation Format — 8-bit. 13