
Zeichencodierungen
Theorie und Übungen

1 Der ASCII-Code

Die Abkürzung ASCII steht für *American Standard Code for Information Interchange*

Ein *Code* ordnet jedem Zeichen aus einer Zeichenmenge eindeutig ein Zeichen oder eine Zeichenfolge einer (möglicherweise anderen) Zeichenmenge zu.

Der ASCII-Code, den es seit etwa 1963 gibt, ordnet den Gross- und Kleinbuchstaben des lateinischen Alphabets, den arabischen Ziffern, einigen Interpunktions- und Sonderzeichen sowie bestimmten Steuerzeichen eindeutig eine Nummer zwischen 0 und 127 zu.

Neben dem ASCII-Code gibt es noch einige andere Codes, die von speziellen Computersystemen gebraucht werden. Beispielsweise den EBCDIC für IBM-Grossrechenanlagen

Die ASCII-Tabelle

| | .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 |

Bei den blau dargestellten Elementen handelt es sich um *Steuerzeichen*, die aus der Zeit der Fernschreiber stammen und heute nicht mehr oder in einem anderem Kontext verwendet werden.

2 Die ISO-8859-X-Standards

Zu Beginn der 80er Jahre des letzten Jahrhunderts kam der Personal Computer auf den Markt. Nun war es auch für Privatpersonen und kleinere Unternehmen erschwinglich, sich einen Computer anzuschaffen.

Mit der weltweiten Verbreitung des PCs mussten weitere zusätzliche Zeichen codiert werden. Diesen Mangel versuchte man u. a. mit den ISO-8859-X-Standards zu beheben.

Nach wie vor verwendete man für die Codierung eines Zeichens auf dem Computer 8 Bit.

Für die vielen Symbole der CJK-Schriften (Chinese-Japanese-Korean) gab es Speziallösungen.

3 Unicode

3.1 Gründe für die Entstehung von Unicode

Ab 1990 wurde die Situation komplizierter, da häufiger Dokumente zwischen verschiedenen Sprachregionen ausgetauscht wurden. Warum?

Computer waren über das Internet verbunden.

1991 wurde die Version 1.0.0 des Unicode-Standards veröffentlicht, um eine Zeichencodierung einzuführen, die einen reibungslosen Datenaustausch über Sprachgrenzen hinweg ermöglichen soll.

Im Jahr 2011 hat der Standard die Version 6 erreicht und umfasst bereits mehr als 100 000 Zeichen.

3.2 Was definiert der Unicode-Standard?

Längerfristig soll für jedes sinntragende Zeichen bzw. Textelement aller bekannten Schriftkulturen und Zeichensysteme ein eindeutiger digitaler Code festgelegt werden. Das bedeutet:

- Jedes Zeichen, das in den Standard aufgenommen wird, erhält *eindeutig* eine „Nummer“. Diese Nummern werden in Ebenen (*planes*) und Blöcke (*blocks*) gruppiert.
- Es wird festgelegt, wie die obigen Nummern digital repräsentiert (=dargestellt) werden. Hier definiert der Standard mehrere Varianten. Dazu gleich mehr.

Ein Zeichen, das eine Nummer bekommen hat, darf nicht mehr unnummeriert werden. Warum?

Eine nachträgliche Änderung würde ein Durcheinander zwischen alten und neuen Versionen des Standards verursachen.

3.3 UTF-32

Für die binäre Darstellung eines Unicode-Zeichens werden 32 Bit (4 Byte) verwendet.

Wie viele Unicode-Zeichen lassen sich damit darstellen?

$2^{32} = 4\,294\,967\,296$ Zeichen

Nachteil: **Grosser Speicherverbrauch**

Vorteil: **Einfache Codierung und Decodierung**

Beispiel: (Lateinischer Buchstabe 'A')

Nummer des Zeichens: $0x41 = 0100|0001$

UTF-32: $00000000\ 00000000\ 00000000\ 01000001$

3.4 UTF-16

Je nach Codenummer werden 16 oder 32 Bit verwendet.

Hat ein Unicode-Zeichen eine Nummer zwischen U+0000 und U+FFFF, dann wird es direkt durch die entsprechende 16 Bit grosse Binärzahl codiert.

Beispiel: (Umlaut 'Ä')

Nummer des Zeichens: **0xC1 = 1100|0001**

UTF-16: **00000000 11000001**

Hat ein Zeichen eine Nummer zwischen U+10000 und U+10FFFF, wird davon die Zahl 0x10000 subtrahiert. Dies ergibt eine Binärzahl zwischen 0x00000 und 0xFFFFF (20 Bits)

Setze die ersten 10 Bits hinter das Präfix *110110* und die zweiten 10 Bits hinter das Präfix *110111*. Da Unicode-Nummern *110110XXXXXXXXXX* und *110111XXXXXXXXXX* für diesen Zweck reserviert sind, gibt es keine Verwechslungen mit den Zeichen, die eine 16-Bit-Codierung haben.

Beispiel: ägyptische Hieroglyphe 

Nummer des Zeichens: **0x1304F**

Subtrahiere 0x10000: **0x0304F**

Binär: **0000|0011|0000|0100|1111**

UTF-16: **110110|0000 0011 00 110111|00 0100 1111**

3.5 UTF-8

Aufbau

Je nach Länge des Zeichencodes werden ein bis vier Bytes benötigt.

- Hat ein Byte die Form **0XXXXXXXX**, so ist es eine ASCII-Code.
- Hat ein Byte die Form **110XXXXX**, so ist es das Startbyte eines Zeichens, das aus 2 Byte besteht.
- Hat ein Byte die Form **1110XXXX**, so ist es das Startbyte eines Zeichens, das aus 3 Byte besteht.
- Hat ein Byte die Form **11110XXX**, so ist es das Startbyte eines Zeichens, das aus 4 Byte besteht.
- Hat ein Byte die Form **10XXXXXX**, so ist es ein Folgebyte, das einen Teil der Zeicheninformation enthält.

Abbildungsbereiche

1 Byte (0XXXXXXXX):

⇒ 7 Bit (0–127)

2 Byte (110XXXXX 10XXXXXX):

⇒ 11 Bit (128–2047)

3 Byte (1110XXXX 10XXXXXX 10XXXXXX):

⇒ 16 Bit (2048–65 537)

4 Byte (11110XXX 10XXXXXX 10XXXXXX 10XXXXXX):

⇒ 21 Bit (65 538–2 097 152)

Vor- und Nachteile

Vorteile:

- Kompatibilität mit den Codierungsschemata ASCII und ISO-Latin.
- Im Mittel weniger Speicherverbrauch durch variable Zeichenlänge.

Nachteile:

- Codieren und decodieren ist aufwändig.
- Sprachen, deren Zeichen grosse Unicode-Nummern haben, werden durch grösseren Speicherbedarf benachteiligt.

3.6 Die Byte Order Mark

Moderne Computerprozessoren fassen mehrere Bytes zu einer Verarbeitungseinheit (*Datenwort*) zusammen.

| Anzahl Bytes | Bezeichnung |
|--------------|---------------------|
| 2 | WORD |
| 4 | DWORD (double word) |
| 8 | LWORD (long word) |

Aus Effizienzgründen verarbeiten nicht alle Prozssortypen die Bytes eines Datenworts in der gleichen Reihenfolge.

Big-Endian-Systeme verarbeiten das höchwertige Byte zuerst:

A3 39 F3 74 → A3 39 F3 74

Little-Endian-Systeme verarbeiten das niederwertigste Byte zuerst:

A3 39 F3 74 → 74 F3 39 A3

Wenn Textdaten im UTF-16 oder UTF-32-Format zwischen Computersystemen ausgetauscht werden, ist die Reihenfolge der Bytes von Bedeutung. (Warum ist dies bei UTF-8 kein Problem?)

Wie kann ein Computer erkennen, ob er bei einem (Unicode-)Text die Bytes in einem Datenwort in umgekehrter Reihenfolge interpretieren muss?

Das Quellsystem speichert am Anfange des Textes die Byte Order Mark (BOM) ab. Im Falle der UTF-16-Codierung handelt es sich um das Unicode-Zeichen U+FEFF.

Wenn ein Computer vom gleichen Verarbeitungstyp dieses Zeichen liest, erkennt er, dass die folgenden Bytes in der „seiner“ Reihenfolge vorliegen. Andernfalls stösst er auf das „illegale“ Unicode-Zeichen U+FFFE und weiss, dass er in jedem Datenwort die Bytes von rechts nach links lesen muss.

4 Übungen

Leerzeichen zwischen hexadezimalen oder binären Zeichenfolgen müssen nicht beachtet werden. Sie dienen der besseren Lesbarkeit.

Wenn nicht anderes steht, kann davon ausgegangen werden, dass bei Multibyte-Codierungen die Bytes in der Big Endian-Reihenfolge angeordnet sind.

Aufgabe 1

Stelle den Text „Wow!“ hexadezimal in den folgenden Codierungen dar:

- (a) UTF-8
- (b) UTF-16
- (c) UTF-32

Aufgabe 1

W=0x57, o=0x6F, w=0x77, !=0x21

- (a) UTF-8: 57 6F 77 21
- (b) UTF-16: 0057 006F 0077 0021
- (c) UTF-32: 00000057 0000006F 00000077 00000021

Aufgabe 2

Handelt es sich bei der hexadezimalen Folge

ef b7 b2

grundsätzlich um eine korrekte UTF-8-Codierung?

Aufgabe 2

| ef | b7 | b2 |
|------------------|------------------|------------------|
| <u>1110</u> 1111 | <u>10</u> 110111 | <u>10</u> 110010 |

Ja, denn das erste Byte zeigt mit der Startsequenz 1110 an, dass es sich um eine 3-Byte-Folge handelt. Die folgenden beiden Bytes beginnen jeweils mit der Sequenz 10, was ebenfalls korrekt ist.

Aufgabe 3

Wie interpretiert ein Programm, das UTF-16-Daten erwartet, den ASCII-Text

Forget it!

Aufgabe 3

| | | | | | | | | |
|----|----|----|---------|----|----|----|----|-------|
| F | o | r | g | e | t | i | t | ! |
| 46 | 6f | 72 | 67 | 65 | 74 | 20 | 69 | 74 21 |
| 襪 | 牧 | 整 | n. def. | 琻 | | | | |

Natürlich muss man die Zeichen in der untersten Zeile weder kennen noch angeben. Wer nach einem bestimmten Unicode-Zeichen sucht, findet die nötigen Informationen unter www.unicode.org/charts/.

Aufgabe 4

Gib die Binärcodierung des Unicode-Zeichens U+2701 in den folgenden Formaten an:

- (a) UTF-16
- (b) UTF-32
- (c) UTF-8

Aufgabe 4

| | |
|----------|----------|
| 27 | 01 |
| 00100111 | 00000001 |

- (a) UTF-16: 00100111 00000001
- (b) UTF-32: 00000000 00000000 00100111 00000001
- (c) UTF-8: Platzbedarf ermitteln:

| 1 Byte | 2 Byte | 3 Byte | 4 Byte |
|----------|----------------------|----------------------------------|--|
| 0xxxxxxx | 110xxxxx 10xxxxxx | 1110xxxx 10xxxxxx 10xxxxxx | 11110xxx 10xxxxxx 10xxxxxx 10xxxxxx |
| 0-7 Bit | 8-11 Bit | 12-16 Bit | 17-21 |

Die Zeichennummer besteht aus 14 Bit. Also sind für UTF-8 drei Bytes nötig: 11100010 10011100 10000001.

Aufgabe 5

Sind die Zeichen, deren Binärdarstellung gegeben ist, im rechts angegebenen Format?

- (a) 11010101 10101111 ASCII
- (b) 01010101 10101111 UTF-16
- (c) 01010101 01110101 UTF-8
- (d) 11010101 01111001 UTF-8

Aufgabe 5

- (a) Nein, das erste Byte beginnt mit einer 1 \Rightarrow ISO-8859-1
- (b) Ja, die Zeichenfolge ist 2 Byte = 16 Bit lang.
- (c) Ja, die Zeichenfolge besteht aus zwei ASCII-Zeichen.
- (d) Nein, das erste Byte zeigt an, dass das UTF-8-Zeichen aus zwei Bytes besteht. Dann müsste das zweite Byte die Form `10xxxxxx` haben.

Aufgabe 6

Die folgenden Hex-Ziffern codieren *ein* Unicode-Symbol in UTF-8:

F0 93 81 A8

- (a) Welche hexadezimale Nummer hat das Symbol?
- (b) Warum reichen 3 Bytes für die UTF-8-Codierung nicht aus?

Nebenbei: Es handelt sich bei dem Symbol um die ägyptische Hieroglyphe



zur Darstellung der Zahl 1 000 000.

Aufgabe 6

- (a) hexadezimal \Rightarrow binär:

| F0 | 93 | 81 | A8 |
|----------------|------------------|------------------|------------------|
| 1111 <u>00</u> | 10 <u>010011</u> | 10 <u>000001</u> | 10 <u>101000</u> |

Die Bits an den unterstrichenen Stellen codieren das Zeichen

00 010011 000001 101000

Teilt man von rechts in Gruppen von 4 Bits auf, so erhält man

| | | | | |
|------|------|------|------|------|
| 0001 | 0011 | 0000 | 0110 | 1000 |
| 1 | 3 | 0 | 6 | 8 |

- (b) UTF-8-Darstellung mit drei Bytes:

1110xxxx 10xxxxxx 10xxxxxx

Also könnten Zeichen bis zu $4 + 6 + 6 = 16$ Bit damit codiert werden.

Die bei (a) bestimmte Bitfolge 0001 0011 0000 0110 1000 besteht jedoch aus 17 Bit (die drei führenden Nullen kann man weglassen). Deshalb ist eine Codierung mit weniger als 4 Byte unmöglich.

5 Lösungen

Aufgabe 1

W=0x57, o=0x6F, w=0x77, !=0x21

- (a) UTF-8: 57 6F 77 21
- (b) UTF-16: 0057 006F 0077 0021
- (c) UTF-32: 00000057 0000006F 00000077 00000021

Aufgabe 2

| ef | b7 | b2 |
|-----------------|-----------------|-----------------|
| <u>11101111</u> | <u>10110111</u> | <u>10110010</u> |

Ja, denn das erste Byte zeigt mit der Startsequenz 1110 an, dass es sich um eine 3-Byte-Folge handelt. Die folgenden beiden Bytes beginnen jeweils mit der Sequenz 10, was ebenfalls korrekt ist.

Aufgabe 3

| F o | r g | e t | i | t ! |
|-------|-------|-------|---------|-------|
| 46 6f | 72 67 | 65 74 | 20 69 | 74 21 |
| 襪 | 牧 | 整 | n. def. | 淑 |

Natürlich muss man die Zeichen in der untersten Zeile weder kennen noch angeben. Wer nach einem bestimmten Unicode-Zeichen sucht, findet die nötigen Informationen unter www.unicode.org/charts/.

Aufgabe 4

| 27 | 01 |
|-----------------|-----------------|
| <u>00100111</u> | <u>00000001</u> |

- (a) UTF-16: 00100111 00000001
- (b) UTF-32: 00000000 00000000 00100111 00000001
- (c) UTF-8: Platzbedarf ermitteln:

| 1 Byte | 2 Byte | 3 Byte | 4 Byte |
|----------|----------|-----------|----------|
| 0xxxxxxx | 110xxxxx | 1110xxxx | 11110xxx |
| | 10xxxxxx | 10xxxxxx | 10xxxxxx |
| | | 10xxxxxx | 10xxxxxx |
| | | | 10xxxxxx |
| 0-7 Bit | 8-11 Bit | 12-16 Bit | 17-21 |

Die Zeichennummer besteht aus 14 Bit. Also sind für UTF-8 drei Bytes nötig: 11100010 10011100 10000001.

Aufgabe 5

- (a) Nein, das erste Byte beginnt mit einer 1 \Rightarrow ISO-8859-1
- (b) Ja, die Zeichenfolge ist 2 Byte = 16 Bit lang.
- (c) Ja, die Zeichenfolge besteht aus zwei ASCII-Zeichen.
- (d) Nein, das erste Byte zeigt an, dass das UTF-8-Zeichen aus zwei Bytes besteht. Dann müsste das zweite Byte die Form $10xxxxxx$ haben.

Aufgabe 6

- (a) hexadezimal \Rightarrow binär:

| | | | |
|----------------|------------------|------------------|------------------|
| F0 | 93 | 81 | A8 |
| 1111 <u>00</u> | 10 <u>010011</u> | 10 <u>000001</u> | 10 <u>101000</u> |

Die Bits an den unterstrichenen Stellen codieren das Zeichen

00 010011 000001 101000

Teilt man von rechts in Gruppen von 4 Bits auf, so erhält man

| | | | | |
|------|------|------|------|------|
| 0001 | 0011 | 0000 | 0110 | 1000 |
| 1 | 3 | 0 | 6 | 8 |

- (b) UTF-8-Darstellung mit drei Bytes:

1110xxxx 10xxxxxx 10xxxxxx

Also könnten Zeichen bis zu $4 + 6 + 6 = 16$ Bit damit codiert werden.

Die bei (a) bestimmte Bitfolge 0001 0011 0000 0110 1000 besteht jedoch aus 17 Bit (die drei führenden Nullen kann man weglassen). Deshalb ist eine Codierung mit weniger als 4 Byte unmöglich.