2.3 Codierung/UTF-8, UTF-16 und UTF-32

2.3.1 Warum UTF-8?

Wenn man versucht, ASCII und ISO-Latin-1 durch UCS-4 und UNICODE abzulösen, trifft man auf mehrere Probleme. Am Beispiel des Buchstabens `A` wird das deutlich: In ASCII wird der Buchstabe `A` durch das Bitmuster 01001001 ausgedrückt. In UCS-4 lautet das entsprechende Bitmuster: 00000000 00000000 00000000 01001001.

- UCS-4-Texte belegen für ein Zeichen vier Byte, das ist viermal so viel wie ASCII-Texte. Das fällt besonders bei der Übertragung großer Textmengen ins Gewicht.
- Alle bisherigen Texte sind ASCII-codiert. Diese Texte können mit UCS4-kompatiblen Werkzeugen nicht einfach so gelesen werden.
- Alle bisherigen Programmierer- und Anwenderwerkzeuge gehen davon aus, dass ein Zeichen mit einem Byte codiert wird. Sie könnten die neuen Texte nicht lesen.
- Viele beliebte Programmiersprachen (z.B. C) benutzen den ASCII-Code Nr. 0 (sein Bitmuster besteht aus lauter Nullen) dazu, das Stringende zu kennzeichnen. Die in diesen Sprachen programmierten Betriebssysteme (Linux, MacOS X, Wind.) sehen ebenfalls diesen Code als Ende z.B. von Dateinamen an. Daher darf dieses Bitmuster nicht in Texten vorkommen.

Gebraucht wird also eine Art Zwischencode, der es erlaubt, alle UCS-4-Zeichen bzw. alle Unicode-Zeichen darzustellen und der trotzdem nicht lauter Nullbytes enthält. Zudem ist es wichtig, dass er aufwärts- und abwärtskompatibel zu ISO-Latin-1 oder zumindest zu ASCII wäre, und dann wäre wünschenswert, wenn häufig benutzte Zeichen kürzer codiert wären als seltener benutzte Zeichen.

2.3.2 Aufbau von UTF-8

UTF-8[1] erfüllt diese Anforderungen sehr gut[2].

In UTF-8 wird jedes Zeichen durch eine Mehr-Byte-Folge (multibyte sequence) dargestellt, die (wie der Name schon sagt) aus einem oder mehreren Bytes besteht.

UTF-8 ist voll kompatibel zu ASCII, denn die UCS-4-Zeichen mit den Nummern 0 bis 127 werden ganz normal in einem Byte mit führender Null gespeichert. Der Buchstabe `A` erhält also das wieder das Bitmuster 01001001.

UTF-8 ist jedoch nicht kompatibel zu ISO-Latin-1, denn die Bitmuster mit führender Eins werden nicht mehr für Umlaute, Akzente oder andere europäische Sonderzeichen verwendet, sondern dienen zur Codierung echter Mehr-Byte-Folgen. Die deutschen Umlaute und das ß müssen also anders dargestellt werden.

Mit Zwei-Byte-Folgen kann man alle Codes bis zur Nummer 2047 speichern, und zwar auf folgende Weise:

- Das erste Byte beginnt mit `110`. Die restlichen fünf Bits sind die fünf oberen Bits der Codenummer.
- Das zweite Byte beginnt mit `10`. Die restlichen sechs Bits sind die sechs unteren Bits der Codenummer.

Zum Beispiel wird das Zeichen Ü, das die Codenummer 220 hat (binär `1101 1100`), so codiert: In das zweite Byte passen die sechs unteren Bits `011100` aus der Codenummer, in das erste Byte kommen die beiden verbleibenden Bits `11`, der Rest wird mit Nullen aufgefüllt (siehe Abbildung 1).

Drei-Byte-Folgen können alle Codes bis zur Nummer 65535 speichern:

- Das erste Byte beginnt mit `1110`. Die restlichen vier Bits sind die fünf obersten Bits der Codenummer.

[1] Abkürzung für: *UCS transformation format* oder *unicode transformation format*

[2] Bei der Programmierung in der Sprache C kann man UTF-8 daher meistens ohne viele Probleme einsetzen, und zwar mit dem normalen Datentyp `char`.

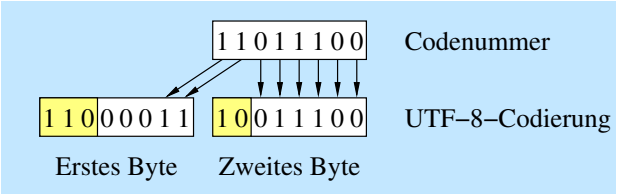

Abbildung 1: Codierung des Zeichens Ü

- Das zweite Byte beginnt mit `10`. Die restlichen sechs Bits sind die sechs mittleren Bits der Codenummer.
- Das dritte Byte beginnt ebenfalls mit `10`. Die restlichen sechs Bits sind die sechs unteren Bits der Codenummer.

Folge-Bytes beginnen also *immer* mit `10`. Abbildung 2 zeigt ein Beispiel für das Ω-Zeichen mit der Codenummer 8486 (binär `0010 0001 0010 0110`).

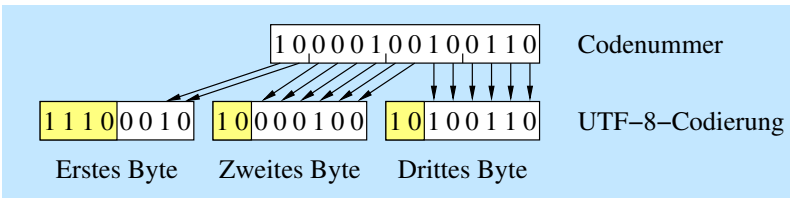

Abbildung 2: Codierung des Zeichens Ω

Weiter kann man nach diesem System definieren:

- Vier-Byte-Folgen, erstes Byte beginnend mit `11110`
- Fünf-Byte-Folgen, erstes Byte beginnend mit `111110`
- Sechs-Byte-Folgen, erstes Byte beginnend mit `1111110`
- Sieben-Byte-Folgen, erstes Byte ist `11111110`

Zur Zeit sind jedoch nur Ein- bis Vier-Byte-Folgen erlaubt; außerdem muss jedes Zeichen so kurz wie möglich codiert werden.

2.3.3 UTF-16 und UTF-32

Die oben genannte Möglichkeit, jedes UCS-4-Zeichen mit vier Byte darzustellen, wird heute oft (analog zu UTF-8) als UTF-32 bezeichnet[3].

Einige Betriebssysteme und Programmiersprachen aus den neunziger Jahren (z.B. Java) haben von Anfang an auf UCS-2 als Zeichensatz gegründet und daher zwei Bytes für jedes Zeichen benutzt. Seitdem UNICODE jedoch mehr als 65536 Codenummern umfasst, muss auch hier für alle Zeichen oberhalb der 65536 bisherigen Zeichen mit Tricks gearbeitet werden: Auch hier braucht man eine Folge aus – in diesem Fall zwei – 16-Bit-Werten, die dann entsprechend codiert und decodiert wird. Diesen Code nennt man UTF-16.

[3]Es ist die einfachste Darstellung. In der Programmiersprache C wird diese Darstellung beim Datentyp `wchar_t` verwendet.