Aufgabe 1

Mit Hilfe der ASCII-Tabelle sucht man sich die hexadezimalen Codes der Zeichen heraus und wandelt dann jede hexadezimale Ziffer (`0`, `1`, ..., `9`, `A`, `B`, ..., `F`) in ein Halbbyte um:

ASCII-Zeichen	W	h	y	?
hexadezimal	0x57	0x68	0x79	0x3F
binär:	0101 0111	0110 1000	0111 1001	0011 1111

Aufgabe 2

`F␣`
`c*`

Aufgabe 3

ef	b7	b2
1110<u>1111</u>	10<u>110111</u>	10<u>110010</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 4

27	01
00100111	00000001

(a) UTF-32: `00000000 00000000 00100111 00000001`

(b) 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: `1110`<u>`0010`</u> `10`<u>`011100`</u> `10`<u>`000001`</u>.

Aufgabe 5

(a) Nein, das erste Bit beginnt mit einer 1

(b) Ja, das erste Bit beginnt mit einer 0

(c) Nein, das erste Byte zeigt an, dass das UTF-8-Zeichen aus zwei Bytes besteht. Das stimmt zwar aber dann müsste das zweite Byte die Form `10xxxxxx` haben.

(d) Ja

(e) Ja

Aufgabe 6

Das 1. Byte beginnt mit 11110. Also besteht das erste Zeichen aus 4 Byte und es müssen noch 3 Byte folgen die jeweils mit 10 beginnen, was korrekt ist.

Das 5. Byte beginnt mit 0. Also handelt es sich beim zweiten Zeichen um ein einzelnes ASCII-Zeichen.

Das 6. Byte beginnt mit 1110. Also besteht das dritte Zeichen aus 3 Byte und es müssen noch 2 Byte folgen die jeweils mit 10 beginnen, was korrekt ist.

Das 9. Byte beginnt wieder mit 11110. Also besteht das vierte Zeichen aus 4 Byte und es müssen noch 3 Byte folgen die jeweils mit 10 beginnen, was stimmt.

Also codiert die Bytefolge 4 Unicode-Zeichen.

Aufgabe 7

(a) hexadezimal ⇒ binär:

F0	93	81	A8
11110000	10010011	10000001	10101000

Die Bits an den unterstrichenen Stellen codieren das Zeichen

`000 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

⇒ U+13068

Die führenden Nullen können weggelassen oder zum Auffüllen auf 4 Bit verwendet werden.

(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.