Dr. Andreas Borchert
Christian Ehrhardt
Institut für Angewandte Informationsverarbeitung

Allgemeine Informatik III (WS 2006/2007)

Abgabetermin 21.Nov 2006

Unicode und UTF-8

ASCII

Bisher wurden Zeichen immer mit Hilfe des ASCII-Codes kodiert. Dabei wird eine Zuordnung zwischen den Zahlen 0 bis 127 (oder 255) und dem zugehörigen Buchstaben getroffen. So entspricht etwa die Zahl 65 dem Buchstaben A und 32 dem Leerzeichen. Jedes Zeichen eines Textes wird dabei einfach als eine 8-Bit Zahl (`char` bzw. `unsigned char` in C) abgespeichert.

Unicode

Das ganze geht solange gut, bis eine Anwendung mehr als 128 (oder 256) verschiedene Zeichen verwenden will. Eventuell auch schon früher, wenn die benötigten Zeichen im ASCII-Code nicht vorkommen. Als Lösung für dieses Problem wurde der sogenannte **Unicode** erfunden. Dabei wird jedem nur denkbaren Zeichen (von lateinischen über griechische bin hin zu chinesischen) ein eindeutiger Integerwert zugeordnet.

UTF-8

Das Problem bei dieser Lösung ist, daß die allermeisten dieser Integerwerte nicht mehr in einem Byte (8-Bit) dargestellt werden können. Zeichen, denen größere Zahlen zugeordnet sind, müssen also mit mehr als einem Byte dargestellt werden. Die vermutlich am meisten verwendete Lösung für dieses Problem ist die UTF-8 Kodierung. Dabei werden zwischen 1 und 4 Byte zur Darstellung einer Integerzahl bzw. eines Unicode-Zeichens verwendet. Im Detail funktioniert das wie folgt:

- Zahlen zwischen 0 und 127 (einschließlich) werden “so wie sie sind” in einem einzigen Byte kodiert. Das heißt, daß klassische ASCII-Texte, die nur die Zahlen 0 bis 127 verwenden, einen gültigen in UTF-8 kodierten Text darstellen. Praktischerweise wurden die ersten 127 Unicode-Zeichen auch so definiert, daß sie den ASCII-Zeichen entsprechen.
- Bei größeren Zahlen wird im ersten Byte angegeben, aus wievielen Zeichen die Kodierung der Zahl besteht. Das geschieht so, daß die Binärdarstellung des ersten Bytes mit genau so vielen Einsen beginnt, wie Bytes für die Zahl benötigt werden. Auf diese Einsen (maximal 4 Stück) folgt in der Binärdarstellung eine 0.
- Alle weiteren Bytes, die zur selben kodierten Zahl gehören, beginnen ihre Binärdarstellung mit 10.
- Die jetzt noch nicht vergebenen Bits aus allen Bytes zusammen bilden die Binärdarstellung der kodierten Zahl.
- Die Anzahl der verwendeten Bytes sollte dabei möglichst minimal sein.

Beispiele

Hier einige Beispiele für korrekte UTF-8 Kodierungen. Die “Nutzdaten” sind dabei fett gedruckt.

Integerwert	Binärdarstellung	Anzahl Bytes	UTF-8 kodiert (binär)
100	1100100	1	0**1100100**
1000	1111101000	2	110**01111** 10**101000**
4000	111110100000	3	1110**0000** 10**111110** 10**100000**
10000	10011100010000	3	1110**0010** 10**011100** 10**010000**

Aufgabe (10 Punkte)

Schreiben Sie ein Programm, das ein Folge von (menschenlesbaren) Integerzahlen zwischen 0 und $2^{21}-1$, wie sie zum Beispiel über die Tastatur eingegeben werden können, in UTF-8 kodiert. Dabei soll jede Zahl genau einem UTF-8 kodierten Unicode-Zeichen entsprechen.
Schreiben Sei ein weiteres Programm, das die Konvertierung in umgekehrter Richtung durchführt.

Hinweise

- Sämtliche Unicode-Zeichen können in Form von verschiedenen PDFs unter http://www.unicode.org/c heruntergeladen werden.
- Chinesische, japanische und koreanische Schriftzeichen (Unified CJK) finden sich z.B. ca. im Bereich zwischen 20000 und 50000 (dezimal).
- Es lohnt sich, die UTF-8 kodierten Zeichen für die Nummern 32 bis 50000 (mit Newlines dazwischen) in eine Datei zu schreiben. Öffnet man diese Datei mit einem Browser und stellt als Kodierung UTF-8 ein, dann kann man feststellen, welche Zeichen der Browser darstellen kann.
- Das Einstellen der Kodierung funktioniert bei `firefox` über View–>Character Encoding–>Unicode(UTF-8)
- Nicht darstellbare Zeichen werden in der Regel durch ein Fragezeichen repräsentiert.