9. Übungsblatt: Programmierpraktikum II (SS 2003)

Abgabe: 7. Juli 2003

Bemerkungen:

- Es sind alle Abgabehinweise auf der Übungswebseite zu beachten!
- Als Programmbibliotheken sind `stdio.h` und `errno.h` zugelassen.

Aufgabe 1: Taschenrechner

(8 Punkte, Abgabedatei: compute.c)

Schreibe ein Linux-Programm, das mit mehreren Argumenten aufgerufen wird. Dazu soll ein (einfacher) Taschenrechner programmiert werden. Wenn das ausführbare Programm z.B. `compute` heißt, dann soll es von der Linux-Konsole in der Form

`compute` *zahl1 operator zahl2*

aufgerufen werden. Dabei sollen `zahl1` und `zahl2` positive ganze Zahlen und `operator` $\in \{+, -, /, \%\}$ sein.[1]

- Entspricht einer der drei Werte nicht dem korrekten Format, so soll das Programm eine Fehlermeldung ausgeben und sich beenden.
- Ist die Eingabe dagegen korrekt, so soll das Programm die beschriebene Rechnung ausführen, das Ergebnis am Bildschirm ausgeben und sich beenden.

Bsp.: Einige Beispiele für das Verhalten des Programmes:

```
> compute 17 + a
Fehlerhafte Eingabe! Korrektes Eingabeformat:
compute zahl1 operator zahl2
> compute 17 / 5
17 / 5 = 3
> compute 17 +
Fehlerhafte Eingabe! Korrektes Eingabeformat:
compute zahl1 operator zahl2
```

[1] Unbepunktete Zusatzaufgabe: Was passiert, wenn man den Operator '*' mit dazu nimmt? Hast du eine Erklärung für das Verhalten des Programmes?

Aufgabe 2: Hamming-Code (Kernfunktionen)

(12 Punkte, Abgabedatei: hamming.c)

Ziel dieser Aufgabe ist es, den (8,4)-Hammingcode aus der Vorlesung *Praktische Informatik II* (Folien 2-60 bis 2-62) zu implementieren. Dazu hatten wir auf Blatt 8 bereits einige Hilfsfunktionen programmiert, die im Folgenden verwendet werden dürfen. Was uns nun noch fehlt, sind die folgenden Kernfunktionen (die wiederum die obigen Hilfsfunktionen benutzen dürfen):

- `int encode(char)`:
  Diese Funktion erhält als Input ein 8-Bit-Datenwort $d = (d_7, \ldots, d_0)$ und liefert das dazugehörige Codewort

  $$w = (d_7, d_6, d_5, d_4, c_3, d_3, d_2, d_1, c_2, d_0, c_1, c_0)$$

  zurück wie auf Folie 2-63 beschrieben.
  Bem.: Beachte, dass c_0 hier als *least significant bit* gespeichert wird!

- `void correct(int*)`:
  Diese Funktion erhält als Input (*Call by Reference*) einen 12-Bit-Wert. Falls dieser einen Fehler enthält (d.h., falls es sich nicht um ein korrektes Codewort handelt), so soll der Fehler korrigiert werden wie auf Folie 2-64 beschrieben.

- `char decode(int)`:
  Diese Funktion erhält als Input ein korrektes 12-Bit-Codewort und liefert den zugehörigen 8-Bit-Datenwert zurück.

Das Hauptprogramm selbst soll wie folgt aussehen:

1. Der Nutzer gibt ein ASCII-Zeichen ein, das Programm ermittelt das zugehörige 12-Bit-Codewort und gibt dessen Binärdarstellung am Bildschirm aus.

2. Der Nutzer gibt einen ganzzahligen Rauschwert r ein (Werte zwischen $0 \leq r \leq 11$ bedeuten, dass das r-te Bit des Codeworts verrauscht wird, alle anderen Eingabewerte bedeuten, dass das Codewort unverändert bleibt). Das Programm gibt die Binärdarstellung des verrauschten Codeworts am Bildschirm aus.

3. Das Programm korrigiert das verrauschte Codewort und gibt die korrekte 12-Bit-Darstellung am Bildschirm aus.

4. Das Programm ermittelt den zugehörigen 8-Bit-Datenwert und gibt dessen ASCII-Darstellung am Bildschirm aus.

Aufgabe 3: Makefile

(5 Punkte, Abgabedateien: s.u.)

Wenn man wie in Aufgabe 2 ein Programm aus Hilfsfunktionen und Kernfunktionen zusammensetzt, dann verwendet man dazu in der Praxis getrennte Dateien und ein Makefile. Gehe dazu wie folgt vor:

- Schreibe die Hilfsfunktionen in eine Datei `divlib.c` und lege eine passende Deklarationsdatei `divlib.h` an.
- Schreibe die Kernfunktionen in eine Datei `corelib.c` und lege eine passende Deklarationsdatei `corelib.h` an.
- Schreibe das Hauptprogramm in eine Datei `code.c`.
- Verlinke diese Dateien durch geeignete `#include`-Direktiven und lege ein Makefile an, so dass der Linux-Befehl `make code` ein ausführbares Programm `code` erzeugt.

Bem.: Wer damit nicht zurecht kommt, der geht bitte in das Tutorium am 3. Juli und läßt sich alles noch einmal ausführlich erklären!