8. Übungsblatt: Programmierpraktikum II (SS 2003)

Abgabe: 30. Juni 2003

Bemerkungen:

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

Aufgabe 1: Binomialkoeffizient

(7 Punkte, Abgabedatei: binomial.c)

Für zwei positive ganze Zahlen n und k, $0 \leq k \leq n$, ist der Binomialkoeffizient $\binom{n}{k}$ definiert wie folgt:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{1 \cdot 2 \cdot \ldots \cdot k} \quad .$$

Da $n!$ aber schon für moderate Werte n extrem groß werden können, ist es nicht sinnvoll, z.B. zuerst $n!$ zu berechnen und dann durch $k!$ bzw. $(n-k)!$ zu dividieren. Stattdessen berechnet man den Binomialkoeffizienten mit der folgenden Rekursion, die die Zwischenergebnisse klein hält:

$$\binom{n}{k} = \frac{n-k+1}{k} \cdot \binom{n}{k-1} \quad ,$$

wobei $\binom{n}{0} = 1$ ist[1].

a) Schreibe eine Funktion, die den Binomialkoeffizienten unter Ausnutzung dieser Äquivalenz **rekursiv** berechnet!

b) Schreibe ein Hauptprogramm, das die Werte n und k einliest, auf Zulässigkeit testet und das Ergebnis $\binom{n}{k}$ ausgibt.

[1] Unbepunktete Zusatzaufgabe: Beweise die Korrektheit dieser Äquivalenz!

Aufgabe 2: Hamming-Code (Hilfsfunktionen)

(8 Punkte, Abgabedatei: hamming.c)

Auf Blatt 8 und 9 soll der 12-Bit-Hamming-Code aus der Vorlesung *Praktische Informatik II* implementiert werden. Dazu benötigen wir zunächst einige Hilfsfunktionen.

- Eine Funktion mit dem Prototypen
  `void displaybits(int)`,
  die einen Integer-Wert als Input erhält und die letzten 12 Bit (*least significant bits, LSB)* seiner Binärdarstellung am Bildschirm ausgibt. So müsste beispielsweise `displaybits(25)` die Bitfolge `000000011001` auf den Bildschirm schreiben.

- Eine Funktion mit dem Prototypen
  `int innerproduct(int, int)`,
  die zwei Integer-Werte a und b als Input erhält und diese als 12-Bit-Werte auffasst. Sie berechnet für $a = (a_{11}, \ldots, a_0)$ und $b = (b_{11}, \ldots, b_0)$ das innere Produkt
  $$c = a_{11}b_{11} \oplus \ldots \oplus a_1 b_1 \oplus a_0 b_0 \quad .$$
  Dabei bezeichnet $\oplus$ das XOR zweier Bitwerte (also die Addition modulo 2). Der Output nimmt daher die Werte 0 oder 1 an.

- Eine Funktion mit dem Prototypen
  `void flipbit(int*, int)`,
  die bei Eingabe von Bitwerten a und b das b-te Bit von a invertiert, falls $0 \leq b \leq 11$ gilt. Andernfalls bleibt a unverändert.
  Bem.: Beachte, dass a per *Call by Reference* übergeben wird!
  Bsp.: Sei `int x = 377`. Dann muss `x` nach Aufruf von `flipbit(&x,6)` den Wert 313 haben, da
  $$\begin{aligned} 000101111001 &= (377)_{10} \\ \oplus\, 000001000000 &= (2^6)_{10} \\ = 000100111001 &= (313)_{10} \end{aligned}$$
  ist.

Schreibe außerdem ein Hauptprogramm, das nacheinander jede der drei Hilfsfunktionen für drei verschiedene Inputs aufruft und zeigt, dass sie die obigen Anforderungen erfüllen.