Prof. Andreas Goerdt
M. Pippig, M. Riedel, U. Schönke

TU Chemnitz
Wintersemester 2003/2004

Algorithmen und Programmierung

4. Übung – Lösungsvorschläge

1. Aufgabe:

   a) Um von einer 2er-Komplementdarstellung mit n Stellen einer Zahl auf deren 2er-Komplementdarstellung mit m Stellen ($m > n$) zu kommen, müssen wir bei positiven Zahlen lediglich vorn $m-n$ Nullen auffüllen, bei negativen Zahlen $m-n$ Einsen.

   Beispiel:

   $n = 4$ Stellen

$$\begin{aligned} 5_{10} &= 0101_2 &= 0101_{2Kpl} \\ -6_{10} &= -0110_2 &= 1010_{2Kpl} \end{aligned}$$

   $m = 8$ Stellen

$$\begin{aligned} 5_{10} &= 00000101_2 &= 00000101_{2Kpl} \\ -6_{10} &= -00000110_2 &= 11111010_{2Kpl} \end{aligned}$$

   b)

$$\begin{aligned} -8_{10} &= -1000_2 &= 1000_{2Kpl} \\ -4_{10} &= -0100_2 &= 1100_{2Kpl} \\ -1_{10} &= -0001_2 &= 1111_{2Kpl} \\ 3_{10} &= 0011_2 &= 0011_{2Kpl} \\ 5_{10} &= 0101_2 &= 0101_{2Kpl} \end{aligned}$$

$$\begin{aligned} 3+(-8) &= 0011_{2Kpl} + 1000_{2Kpl} &= 1011_{2Kpl} &= 0101_2 &= -5 \\ (-4)+(-1) &= 1100_{2Kpl} + 1111_{2Kpl} &= 11011_{2Kpl} => 1011_{2Kpl} &= 0101_2 &= -5 \\ (-8)+(-1) &= 1000_{2Kpl} + 1111_{2Kpl} &= 10111_{2Kpl} => 0111_{2Kpl} &= 0111_2 &= 7 \end{aligned}$$

   Die letzte Addition liefert ein falsches Ergebnis, da $(-8)+(-1) = (-9)$ außerhalb des durch das 2er-Komplement mit 4 Stellen darstellbaren Zahlenbereiches liegt.

   c) An den Überträgen allein kann man das nicht erkennen.

   Es gibt zwei Fälle, wo Fehler auftreten:

- Addition der Komplemente von **2 positiven Zahlen** (Ziffer an Position $n-1$ beide Male 0) und es existiert ein **Übertrag** in die $(n-1)$-te Stelle
  $\Rightarrow$ Fehler, da die $(n-1)$-te Stelle zu 1 und damit das Ergebnis negativ wird (Komplement!).
- Addition der Komplemente von **2 negativen Zahlen** (Ziffer an Position $n-1$ beide Male 1) und es existiert **kein Übertrag** in die $(n-1)$-te Stelle
  $\Rightarrow$ Fehler, da die $(n-1)$-te Stelle durch die Addition zu 0 und damit das Ergebnis positiv wird.

In allen anderen Fällen kann zwar ein Übertrag in die n-te Stelle auftreten, dies signalisiert jedoch keinen Fehler (s. Bsp. oben).

Allgemein kann man sagen, dass

$$0 \leq x_{n-1} + y_{n-1} - c_{n-1} \leq 1$$

gelten muss, damit die Addition fehlerfrei ist.

d) Wir erzeugen -7 in Zweier-Komplementdarstellung auf 4 Stellen durch Divisionsmethode

$$\begin{aligned} -7 &= (-4)\cdot 2 + 1 \\ -4 &= (-2)\cdot 2 + 0 \\ -2 &= (-1)\cdot 2 + 0 \\ -1 &= (-1)\cdot 2 + 1 \end{aligned}$$

Man erhält offensichtlich wirklich die 2er-Komplementdarstellung $-7_{10} = 1001_{2Kpl}$.

Zufall? Nein! Begründung:

$$\begin{aligned} -7 &= \underbrace{-4}\cdot 2 + 1 \\ &= (\underbrace{-2}\cdot 2 + 0)\cdot 2 + 1 \\ &= ((\underbrace{-1}\cdot 2 + 0)\cdot 2 + 0)\cdot 2 + 1 \\ &= (((-1\cdot 2 + 1)\cdot 2 + 0)\cdot 2 + 0)\cdot 2 + 1 \end{aligned}$$

Ausmultiplizieren liefert:

$$\begin{aligned} -7 &= -1\cdot 2^4 + 1\cdot 2^3 + 0\cdot 2^2 + 0\cdot 2^1 + 1\cdot 2^0 \\ &= -16 + 8 + 0 + 0 + 1 \end{aligned}$$

Wir addieren nun auf beiden Seiten der Gleichung 2^4, also 16:

$$-7 + 16 = 8 + 0 + 0 + 1$$

Daraus erkennen wir die Analogie zur Definition des 2er-Komplementes:

$$\begin{aligned} \mathrm{MOD}(a, 2^n) &= 2^n + a \quad \text{für} \quad -2^n \leq a < 0 \\ \mathrm{MOD}(a, 2^n) &= a \quad\quad\;\; \text{für} \quad 0 \leq a < 2^n \end{aligned}$$

2. Aufgabe:

   Wenn wir eine in einem Zahlensystem dargestellte Zahl in ein anderes System umwandeln wollen, dann müssen wir systematisch durch die Basis des Zielsystems dividieren. Wollen wir also $(z_{n-1}...z_0)_2$ in $(x_{m-1}...x_0)_{2^a}$ umwandeln, müssen wir durch 2^a dividieren. Bei einer Binärzahl entspricht das dem Abschneiden der letzten a Stellen, der verbleibende Rest bildet die jeweilige Ziffer im Zielsystem. Demzufolge kann man die Ziffern der Ausgangsdarstellung gleich (von rechts beginnend!) in Gruppen von a Stellen aufteilen und diese jeweils direkt ins Zielsystem umwandeln.

   Beispiel: Wandeln die Zahl 15077_{10} vom Binärsystem (also 11101011100101_2) ins Hexadezimalsystem (Basis $2^4 = 16$) um:

   $$\underbrace{(00)11}_{3} \quad \underbrace{1010}_{A} \quad \underbrace{1110}_{E} \quad \underbrace{0101}_{5}$$

3. Aufgabe:

   b) Es gibt keinen vorgefertigten Mechanismus in Java, um derartige Überläufe abzufangen. Wenn in Ihrem Programm ein solcher Fehler auftreten und zu Problemen führen könnte, müssen Sie sich selbst um das vorherige Testen kümmern.

4. Aufgabe:

   Induktionsanfang

   $n = 0$:

   Es ist:

   $a^0 = 1$ und $\frac{1-a^{0+1}}{1-a} = \frac{1-a}{1-a} = 1$

   Induktionsschritt

   Induktionsvoraussetzung: $n = k$ und es gelte $\sum_{i=0}^{k} a^i = \frac{1-a^{k+1}}{1-a}$

   Setzen nun $n = k+1$:

   $$\begin{aligned} \sum_{i=0}^{k+1} a^i &= \sum_{i=0}^{k} a^i + a^{k+1} &= \frac{1-a^{k+1}}{1-a} + a^{k+1} \\ &&= \frac{1-a^{k+1}+a^{k+1}(1-a)}{1-a} \\ &&= \frac{1-a^{k+1}+a^{k+1}-a^{k+1}\cdot a}{1-a} \\ &&= \frac{1+a^{k+2}}{1-a} \end{aligned}$$

   Gilt die Formel also für $n = k$, so gilt sie auch für $n = k+1$.

   Damit wurde allgemein $\sum_{i=0}^{k} a^i = \frac{1-a^{k+1}}{1-a}$ für alle $n \geq 0$ gezeigt. a ist dabei beliebig, jedoch ungleich 1, da sonst Division durch Null erfolgt (im Nenner steht dann 1-1).

5. Aufgabe:

   Um alle Vielfachen von 2 (außer 2 selbst) zu überspringen, erhöhen wir d immer gleich um zwei. Dabei müssen wir jedoch mit einer ungeraden Zahl (also 3) beginnen, weshalb die 2 am Anfang eine Sonderbehandlung bekommt. Die Vielfachen von 3 und 5 schließen wir aus, indem wir d solange erhöhen, bis bei der ganzzahligen Division sowohl durch 3 als auch durch 5 ein Rest bleibt.

```
import Prog1Tools.IOTools;

public class FaktorBesser {

  public static void main(String[] args) {

    long a ,d;
    a =  IOTools.readLong("Einlesen␣eines␣langen␣a␣>=␣2:␣");
    System.out.println(a +"␣soll␣zerlegt␣werden.");
    System.out.println();
    d = 2;

    // Sonderbehandlung Primfaktor 2
    while (a % d == 0){
        System.out.println( d + "␣ist␣Primfaktor");
        a = a/ d;
      }
    d++;

    // hoehere Primfaktoren
    while (d * d <= a) {
      while (a % d == 0){
        System.out.println( d + "␣ist␣Primfaktor");
        a = a/ d;
      }
      do {
        d=d+2; // nur ungerade Zahlen
      } while ((d%3==0) || ((d%5==0)&&(d>5)));
              //keine Vielfachen von 3 oder 5
    }

    if (a>1) {
      System.out.println(a + "␣ist␣Primfaktor");
    }
  }
}
```