Praktische Informatik 3 WS 06/07

1. Übungsblatt

Ausgabe: 31.10.06

Bearbeitungszeit: Zwei Wochen

Christoph Lüth <cxl>
Matthias Berger <tokio>
Klaus Hartke <hartke>
Cui Jian <ken>
Friederike Jolk <rikej>
Christian Maeder <maeder>
Diedrich Wolter <dwolter>

1 *Schiefer Wurf* *3 Punkte*

Ein mit der Geschwindigkeit v_0 und dem Winkel ϕ von der Erdoberfläche geworfener (oder getretener) Ball wird (ohne Berücksichtigung des Luftwiderstandes) in der Entfernung

$$d = \frac{v_0^2 \sin(2\phi)}{g} \tag{1}$$

wieder auf der Erde aufkommen (mit der Fallbeschleunigung $g = 9.80665 \frac{m}{s^2}$). Implementieren Sie eine Funktion

```
throw :: Double-> Double-> Double
```

welche aus der Anfangsgeschwindigkeit (erstes Argument) und dem Abwurfwinkel (zweites Argument) mit der Formel (1) die Wurfweite berechnet. Der Winkel soll in Grad angegeben werden, für die Formel (1) muss also noch die Umrechnung in Radiant durchgeführt werden.

2 *Schneller!* *7 Punkte*

In der folgenden Aufgabe wird die Potenz a hoch m modulo einer Basis n benötigt. Daher betrachten wir zuerst ein Verfahren, welches a hoch m schnell berechnet. Um die m-te Potenz einer ganzen Zahl a zu berechnen, können wir entweder a m-mal mit sich selbst multiplizieren (was ziemlich lange dauern kann), oder folgenden einfachen Sachverhalt nutzen:

$$a^{2m} = a^m \cdot a^m \tag{2}$$
$$a^{2m+1} = a^m \cdot a^m \cdot a \tag{3}$$

Denselben Trick können wir auch benutzen, um die m-te Potenz modulo n zu berechnen; dabei ist die Multiplikation durch die Multiplikation modulo n (d.h. $x \cdot y = (x\,y) \bmod n$) zu ersetzen.

Implementieren Sie unter Nutzung dieser Tatsachen eine Funktion `modexp a m n`

```
modexp :: Integer-> Integer-> Integer-> Integer
```

welche $a^m \bmod n$ berechnet.

Hinweis: Bringen Sie dazu die Gleichungen (2, 3) in eine rekursive Form mit einer Fallunterscheidung; nutzen Sie die Funktionen `even, odd :: Integer-> Bool`.

3 *Prim oder nicht prim?* *10 Punkte*

Primzahlen, besonders große, werden ja immer gebraucht, unter anderem für kryptographische Zwecke (zur Schlüsselgenerierung) und in Platin gefasst als attraktives Schmuckstück für den Liebhaber.

Um zu testen, ob eine Zahl m eine Primzahl ist, kann man für alle Zahlen von 2 bis $\lfloor\sqrt{m}\rfloor$ testen, ob sie m teilen, was für große Zahlen sehr aufwändig ist. Die Firma Fermat PrimeNumbers, Inc. ist jetzt mit einem neuen Verfahren auf dem Markt, welches wesentlich weniger Aufwand benötigt. Das Verfahren beruht auf dem "kleinen" Theorem von Pierre de Fermat (dem Urgroßvater des Firmeninhabers): Wenn p eine Primzahl ist, dann ist für alle mit p teilerfremden a

$$a^{p-1} \bmod p = 1 \tag{4}$$

Das Verfahren prüft einfach, ob Gleichung (4) für $a = 2$ gilt. Ist dies nicht der Fall, kann p keine Primzahl sein, aber der (logisch eigentlich nicht zulässige) Umkehrschluss trifft überraschend oft auch zu: wenn die Gleichung gilt, ist p (wahrscheinlich) eine Primzahl.

1. Implementieren Sie eine Funktion

   ```
   fastprime :: Integer-> Bool
   ```

   die prüft, ob das Argument nach dem Fermatschen Verfahren eine Primzahl ist.

2. Implementieren Sie eine Funktion

   ```
   fastprimetest :: Integer-> Integer-> Integer
   ```

   `fastprimetest k l` soll für alle Zahlen von k bis l prüfen, ob `fastprime` fälschlicherweise `True` zurückgibt. Ist das der Fall, wird die kleinste solche Zahl zurückgegeben; ist das nicht der Fall, wird 0 zurückgegeben.

   Für wieviele Zahlen von 1 bis 10.000 liefert `fastprime` ein falsches Ergebnis?