Universität Bremen

Praktische Informatik 3 WS 22/23

Christoph Lüth
Raphael Baass
Thomas Barkowsky
Tede von Knorre
Alexander Krug
Tarek Soliman

1. Übungsblatt

Ausgabe: 24.10.22
Abgabe: 30.10.22

1.1 *Quadratisch, Praktisch, Gut.* *5 Punkte*

In dieser ersten Übungsaufgabe wollen wir abstrakte ASCII-Kunst produzieren. Etwas spezifischer, wir wollen eine Funktion implementieren, welche eine quadratische Box gegebener Länge zurückgibt:

```
box 5 '*'  ⇝  "*****\n*␣␣␣*\n*␣␣␣*\n*␣␣␣*\n*****\n"
putStr (box 5 '*')  ⇝
*****
*   *
*   *
*   *
*****
```

Wir erzeugen diese Quadrate in zwei Schritten:

1. Schreiben Sie zuerst eine Funktion `line c n`, welche eine Zeichenkette der Länge `n` aus Zeichen `c` erzeugt:

   ```
   line :: Int→ Char→ String
   line 4 ' '  ⇝  "␣␣␣␣"
   line 0 '+'  ⇝  ""
   ```

   Ist das erste Argument negativ, wird die leere Zeichenkette zurücgegeben.

2. Implementieren Sie jetzt die Funktion `box n c`, welche ein Quadrat der Kantenlänge `n` aus Zeichen `c` zurückgibt. Das Quadrat wird schrittweise konstruiert:

   (a) Zuerst eine Zeile der Länge `n` aus Zeichen `c`,

   (b) verkettet mit `n-2` Zeichenketten, die aus dem Zeichen `c`, gefolgt von `n-2` Leerzeichen, gefolgt von dem Zeichen `c` bestehen,

   (c) und abschließend noch einmal eine Zeichenkette der Länge `n` aus Zeichen `c`.

   Hier das Schema für `box 5 '*'`:

   ```
   "*****\n" ++ ("*␣␣␣*\n" ++ "*␣␣␣*\n" ++ "*␣␣␣*\n") ++ "*****\n"
   Schritt 1               Schritt 2                    Schritt 3
   ```

Beachten Sie, dass alle Zeilen mit einem Zeilenvorschub (`"\n"`) abgeschlossen werden. Für $n \leq 0$ soll die leere Zeichenkette zurückgegeben werden.

Hinweise:

1. Die Funktion `rep` aus der Vorlesung ist hier sehr nützlich, und kann verwendet werden (einfach in die Quelldatei kopieren); sie muss allerdingsfür den Fall $n < 0$ angepasst werden.
2. Um aus einem Zeichen `c` eine Zeichenkette zu machen, hängen Sie das Zeichen vorne an die leere Zeichenkette `c:""`.

1.2 *Perfekt!* *5 Punkte*

Eine natürliche Zahl $n \in \mathbb{N}$ heißt *perfekt*, wenn die Summe der Teiler von n (einschließlich 1, aber ausschließlich n selber) genau wieder n ergibt. Ein Beispiel für eine perfekte Zahl ist 28: die Teiler sind $1+2+4+7+14 = 28$; ein Gegenbeispiel ist 45 mit $1+3+5+9+15 = 33$.

Wir wollen eine Funktion

```
perfect :: Integer→ Bool
```

implementieren, die prüft, ob eine Zahl perfekt ist. Dazu gehen wir in mehreren Schritten vor:

1. Zuerst schreiben wir eine Funktion `divides n m`, die prüft, ob `n` ein Teiler von `m` ist; das ist der Fall, wenn der Rest von `m` geteilt durch `n` 0 ist.

   ```
   divides :: Integer→ Integer→ Bool
   divides 3 7  ⇝  False
   divides 7 35  ⇝  True
   ```

2. Dann schreiben wir eine Funktion `sumOfDivisors`, welche die Summe der Teiler von `n` berechnet:

   ```
   sumOfDivisors :: Integer→ Integer
   sumOfDivisors 3  ⇝  1
   sumOfDivisors 45  ⇝  33
   sumOfDivisors 28  ⇝  28
   ```

   Dazu müssen wir alle Zahlen von `1` bis `n-1` prüfen, ob sie ein Teiler von `n` sind; ist das der Fall, werden sie zum Ergebnis addiert, ansonsten nicht. Das machen wir, indem wir eine Hilfsfunktion `sumDiv2 n m` implementieren, wobei `m` (rekursiv!) die Werte von `1` bis `n-1` durchläuft.

3. Mit `sumOfDivisors` können wir jetzt die gewünschte Funktion `perfect` leicht implementieren.

Die ersten perfekten Zahlen sind übrigens $6, 28, 496$ und 8128. Es ist unbekannt, wie viele perfekte Zahlen es gibt, oder ob es auch ungerade perfekte Zahlen gibt.[1] Im Gegensatz zu Primzahlen haben sich für perfekte Zahlen noch keine Anwendungen in der Kryptologie gefunden, aber weil sie vergleichsweise selten sind, werden sie unter Kennern zu Höchstpreisen als NFTs gehandelt.

[1] `https://www.britannica.com/science/perfect-number`