Universität Bremen

Praktische Informatik 3 WS 2023/24

Thomas Barkowsky
Tede von Knorre
Yaroslav Purgin
Tim Sperling

1. Übungsblatt

Ausgabe: 23.10.2023
Abgabe: 30.10.2023

1.1 *Text im Kasten* *4 Punkte*

In dieser ersten Übungsaufgabe wollen wir eine Funktion textInFrame schreiben, die einen vorgegebenen String eingerahmt in ein ebenfalls übergebenes Zeichen zurückgibt:

```
textInFrame "Hello␣Haskell!" '*' ⇝ "*****************\n*␣Hello␣Haskell!␣*\n*****************\n"
```

Einen so generierten String können wir mit der Funktion putStr im Terminal ausgeben lassen:

```
putStr (textInFrame "Don't␣Panic!" '#') ⇝
################
# Don't Panic! #
################
```

Hinweise:

1. Die vordefinierten Haskell-Funktionen length und replicate sind für die Lösung dieser Aufgabe sehr nützlich.
2. Alle drei Zeilen der Textbox sollen mit einem Zeilenvorschub ("\n") abgeschlossen werden.

1.2 *Primzahlzwillinge* *6 Punkte*

Primzahlen sind natürliche Zahlen größer als 1, die ganzzahlig nur durch 1 und sich selbst teilbar sind. Primzahlzwillinge sind (aufsteigend geordnete) Paare von Primzahlen, die voneinander den Abstand 2 haben, wie beispielsweise (5, 7) oder (17, 19). In dieser Aufgabe wollen wir eine Funktion isPrimeTwin schreiben, die überprüft, ob es sich bei einem vorgegebenen Zahlenpaar um Primzahlzwillinge handelt oder nicht:

```
isPrimeTwin (17,19) ⇝ True
isPrimeTwin (19,21) ⇝ False
isPrimeTwin (19,23) ⇝ False
```

Wir nähern uns dem Problem schrittweise. Zuerst schreiben wir eine Funktion divides n m, die prüft, ob n ein Teiler von m ist. Das ist genau dann der Fall, wenn der Rest von m geteilt durch n 0 ist:

```
divides 3 7 ⇝ False
divides 7 35 ⇝ True
```

Die Haskell-Funktion mod ist hierfür sehr nützlich.

Die Funktion divides können wir dann zur Definition einer Funktion

```
isPrime :: Int → Bool
```

verwenden, die prüft, ob eine übergebene Zahl eine Primzahl ist oder nicht. Für diese Überprüfung untersuchen wir für alle Zahlen von 2 bis zu der Hälfte der zu untersuchenden Zahl, ob sich darunter ein Teiler befindet. Wird kein Teiler gefunden, handelt es sich um eine Primzahl. Allerdings müssen wir diese Überprüfung rekursiv durchführen, weshalb wir eine Hilfsfunktion benötigen, die neben der zu überprüfenden Zahl auch den aktuell betrachteten (potentiellen) Teiler erhält:

```
isPrimeHelp :: Int → Int → Bool
```

Hierbei ist der erste übergebene Parameter die zu prüfende Zahl; der zweite Parameter ist der im aktuellen Rekursionsschritt betrachtete Teiler.

Diese Hilfsfunktion rufen wir dann in der Funktion isPrime auf und lassen die Prüfung rekursiv mit den Teilern von 2 bis zur Hälfte der zu untersuchenden Zahl laufen (oder auch in der umgekehrten Richtung, das geht natürlich auch). Schließlich können wir die Funktion

```
isPrimeTwin :: (Int, Int) → Bool
```

definieren, die die oben beschriebenen Prüfung durchführt. Wenn ihr alles richtig gemacht habt, könnt ihr mit der Funktion

```
allPrimeTwins :: [(Int, Int)]
allPrimeTwins = filter isPrimeTwin [(a, a+2)| a ← [3,5..]]
```

alle Primzahlzwillinge berechnen lassen und euch beispielsweise die ersten 10 davon ausgeben lassen:

```
take 10 allPrimeTwins ⇝
[(3,5),(5,7),(11,13),(17,19),(29,31),(41,43),(59,61),(71,73),(101,103),(107,109)]
```

(Ihr braucht im Moment noch nicht zu verstehen, was die obige Funktion genau tut, das kommt später...)