Philipps Universität Marburg

Übungen zu „Konzepte von Programmiersprachen“, WS 2010/11

Prof. Dr. R. Loogen · Fachbereich Mathematik und Informatik · Marburg

Nr. 1, Abgabe: Dienstag, 26. Oktober 2010 vor der Vorlesung

Die Lösungen müssen schriftlich Programme zusätzlich per E-Mail an den Tutor oder die Tutorin abgegeben werden. Die Abgabe ist in Gruppen bis zu zwei Personen erlaubt.

A. Hausaufgaben

1. Vergleich von Argumenten — 5 Punkte

   (a) Geben Sie Definitionen der folgenden Funktionen an: / 3

   ```
   howManyOfTwoEqual :: Int -> Int -> Int
   howManyEqual      :: Int -> Int -> Int -> Int
   allDifferent      :: Int -> Int -> Int -> Bool
   ```

   `howManyEqual` und `howManyOfTwoEqual` zählen, wieviele der gegebenen Funktionsargumente gleich sind. `allDifferent` liefert den Wert `True` zurück, wenn alle drei Argumente verschieden sind.

   Der Operator `/=` liefert beim Aufruf `m /= n` den Wert `True`, wenn `m` ungleich `n`.

   (b) Geben Sie eine alternative Definition `howManyEqual'` an, die die Funktionen `allEqual` und `allDifferent` verwendet. `allEqual` sei wie folgt definiert: / 1

   ```
   allEqual :: Int -> Int -> Int -> Bool
   allEqual n m p = (n == m) && (m == p)
   ```

   (c) Geben Sie Testdaten an, die zeigen, dass die folgende alternative Definition von `allEqual` falsch ist. Wo liegt das Problem? / 1

   ```
   allEqual'       :: Int -> Int -> Int -> Bool
   allEqual' n m p =  ((n+m+p) == 3*p)
   ```

2. Bildtransformationen — 7 Punkte

   Das Modul `Picture` in der auf der Vorlesungsseite bereitgestellten Datei `picture.hs` stellt folgende Funktionen für einfache Bildtransformationen zur Verfügung:

   ```
   module Picture where type Picture = ...
     printPic            :: Picture -> IO()    -- Bildschirmausgabe von Bildern
     flipH, flipV, flipD :: Picture -> Picture
                         -- Spiegelung horizontal, vertikal, diagonal
     invertColour        :: Picture -> Picture -- Farbinversion
     superimpose, above, aside  :: Picture -> Picture -> Picture
                         -- Bildanordnung ueber-, unter-, nebeneinander
     white, lambda       :: Picture            -- weisses Rechteck und Lambda
   ```

   Durch Angabe von `import Picture` am Beginn Ihres Programms können Sie das Modul importieren und die Funktionen verwenden.

   (a) Definieren Sie eine Funktion / 3

```
chessboard :: Int -> Picture -> Picture
```

die zu einer positiven Zahl `n` und einem Bild ein Schachbrett erzeugt, in dem das Bild in jeder Dimension `n`-mal abwechselnd positiv und negativ, d.h. mit invertierten Farben, wiederholt wird. `chessboard 8 white` definiert also ein normales Schachbrett.

(b)
```
##........
##........
..##......
..##......
....##....
....##....
......##..
......##..
........##
........##
```

Definieren Sie Funktionen `diag` und `cross` mit dem Typ `Int -> Picture`, die zu einer ganzen Zahl n eine $n \times n$ Matrix mit weißen und schwarzen Rechtecken bilden, in der die schwarzen Rechtecke diagonal von links oben nach rechts unten angeordnet sind (links) bzw. in der die schwarzen Rechtecke ein Kreuz bilden, das diagonal entgegengesetzte Ecken verbindet (rechts).

```
##......##
##......##
..##..##..
..##..##..
....##....
....##....
..##..##..
..##..##..
##......##
##......##
```

/ 4

B. Mündliche Aufgaben (für die erste Übungsstunde)

M1 Reduktion

Zusätzlich zu der in Hausaufgabe 1(b) definierten Funktion `allEqual` seien die folgenden Funktionsdefinitionen gegeben:

```
simple       :: Int -> Int -> Int -> Int
simple a b c =  a * (b+c)

square   :: Int -> Int
square a =  a * a
```

(a) Bestimmen Sie alle Redexe im folgenden Ausdruck:

```
allEqual (simple (2-1) (square (2+1)) (square (2+2)))
         (square (3+2)) (5*(3+2))
```

(b) Führen Sie eine ausführliche Auswertung des Ausdrucks durch.

M2 Bildtransformationen

Diese Aufgabe bezieht sich auf das Modul `Picture` aus Hausaufgabe 2.

(a) Definieren Sie auf zwei verschiedene Weisen ein schwarzes Rechteck

```
black :: Picture.
```

(b) Geben Sie zwei verschiedene Möglichkeiten an, ein Bild der folgenden Gestalt zu erzeugen:

```
......######
......######
......######
######......
######......
######......
```

(c) Definieren Sie eine Funktion `picCol :: Int -> Picture -> Picture`, die ein Bild so oft untereinander anordnet wie durch den ersten Parameter angegeben.

(d) Definieren Sie eine Funktion `rotate90 :: Picture -> Picture`, die ein Bild um 90 Grad nach rechts dreht.