Universität Bremen

Praktische Informatik 3 WS 20/21

Christoph Lüth
Thomas Barkowsky
Tobias Brandt
Alexander Krug
Tarek Soliman
Robert Sachtleben

4. Übungsblatt

Ausgabe: 30.11.2020
Abgabe: 07.12.2020 12:00

Vorbemerkung: verzichten Sie in diesem Übungsblatt wenn irgendwie möglich auf die Definition rekursiver Funktionen, und nutzen Sie Funktionen höherer Ordnung wie in der Vorlesung vorgestellt.

4.1 *BIG LETTERS* *7 Punkte*

Ihre älteren Verwandten beklagen sich, dass Ihre Weihnachtskarten immer so klein geschrieben sind, so dass man sie schlecht lesen könne. Um der drohenden Enterbung vorzubeugen wollen wir jetzt ein Programm schreiben, dass im Stil des Unix-Programms `banner` RIESIG schreiben kann.

Dieses Programm hat drei Komponenten:

1. Die großen Zeichen sind in einer Art Bitmap kodiert. Wir müssen also zuerst Zahlen in Folgen von Bits kodieren und anders herum.

   Implementieren Sie dazu zwei Funktionen: die erste konvertiert eine positive ganze Zahl in eine Liste von Wahrheitswerten (wobei der erste Parameter die Wortbreite, d.h. die Anzahl Bits und damit die Länge der Ergebnisliste angibt), und die zweite konvertiert eine Liste von Wahrheitswerten in eine ganze Zahl. Die Wahrheitswerte sind LSB-0 geordnet, d.h. das kleinste Bit zuerst:

   ```
   decode :: Int→ Integer→ [Bool]
   encode :: [Bool]→ Integer
   decode 8 79 ⇝ [True,True,True,True,False,False,True,False]
   encode [False, True, True, False, True, False, True] ⇝ 86
   ```

   1 Punkt

2. Die Daten für die großen Buchstaben sind einer Tabelle `charMap` definiert (in der Vorlage schon enthalten). Die `charMap` besteht aus einer Assoziationsliste von Zeichen und einer Liste von acht ganzen Zahlen. Jeder dieser Zahlen kodiert dann acht Bits:

   ```
   charMap :: [(Char, [Integer])]
   ```

   Die Zeichen sind also 8x8 Zeichen groß. Der Eintrag für `'A'` lautet beispielsweise

   ```
   ('A', [8, 20, 34, 65, 127, 65, 65, 0])
   ```

   Die Zahlen als Binärzahlen (in LSB-0-Ordnung) sind

   ```
   [00010000, 00101000, 01000100, 10000010, 11111110, 10000010, 10000010, 00000000]
   ```

   und wenn wir das untereinander schreiben

   ```
   00010000
   00101000
   01000100
   10000010
   11111110
   10000010
   10000010
   00000000
   ```

   Wenn wir jetzt die Nullen durch ein Leerzeichen, und die Einsen durch einen beliebiges Zeichen, bspw. $*$, ersetzen, erhalten wir

```
   *
  * *
 *   *
*     *
*******
*     *
*     *
```

Um ein einzelnes Zeichen `c` zu schreiben müssen wir also:

(a) Die entsprechende Zeile in der `charMap` finden;

(b) jede der Zahlen in diesem Eintrag in Binärdarstellung konvertieren; und

(c) in der Binärdarstellung jedes `False` durch `' '` und jedes `True` durch ein Zeichen `o` ersetzen.

Zusammengesetzt ergibt das eine Funktion

```
banner1 :: Char→ Char→ [String]
```

wobei `o` das erste und `c` das zweite Argument ist:

```
banner1 '∗' 'A' ⇝
[ "␣␣␣*␣␣␣␣"
, "␣␣*␣*␣␣␣"
, "␣*␣␣␣*␣␣"
, "*␣␣␣␣␣*␣"
, "*******␣"
, "*␣␣␣␣␣*␣"
, "*␣␣␣␣␣*␣"
, "␣␣␣␣␣␣␣␣"]
```

Nicht in der `charMap` enthaltene Zeichen sollten entweder gar nicht, oder durch eine definierte Fehlerausgabe (8x8 Fragezeichen) dargestellt werden.

3 Punkte

3. Jetzt wollen wir natürlich längere Botschaften als einzelne Zeichen ausgeben. Schreiben Sie dazu eine Funktion welche einen String in 8x8-Großbuchstaben mit dem übergebenen Zeichen darstellt:

```
banner :: Char→ String→ String
putStr (banner '#' "Happy?") ⇝
#     #                                    ####
#     #                                   #    #
#     #  ####   #####   #####    #   #        #
#######      #   #   #   #   #   #   #     ###
#     #  #####   #   #   #   #   #   #     #
#     # #    #   ####    ####     #  #
#     #  #### #  #       #      #  ##      #
                ##      ##       ###
```

Wir implementieren die Funktion `banner` in vier Schritten:

(a) Jedes Zeichen wird mit `banner1` dargestellt;

(b) die resultierende Liste von Listen von Zeilen pro Zeichen (erst alle Zeilen des ersten Zeichens, dann alle Zeilen des zweiten Zeichens usw.) wird in eine Liste von Listen von Zeilen umgewandelt (d.h. zuerst alle ersten Zeilen, dann alle zweiten Zeilen usw. bis zu der Liste aller achten Zeilen);

(c) dann jeweils die Listen der Zeilen zusammenfügen;

(d) Zum Schluss die Liste der Zeilen zu einem String zusammenfügen und dabei einen Zeilenvorschub einfügen.

Schreiben Sie sich für jeden dieser Schritte die Signatur auf. Jeder dieser Schritte besteht nur aus ein oder zwei vordefinierten Funktionen. `banner` ist dann idealerweise die Komposition dieser Funktionen.

3 Punkte

Hinweise:

1. Das Modul `Data.Bits` bietet die nötigen Funktionen zur Manipulation von Bits, wie wir sie aus Java und C kennen: `(.&.)`, `(.|.)` und `shiftL`.
2. Folgende vordefinierte Funktionen können hilfreich sein:

   ```
   lookup    :: Eq α ⇒ α → [(α, β)] → Maybe β
   transpose :: [[α]] → [[α]]
   unlines   :: [String] → String
   ```

   `transpose` muss aus `Data.List` importiert werden, und vertauscht die Zeilen und Spalten in einer als Liste von Listen repräsentierten Matrix; die anderen beiden können so benutzt werden.

4.2 *Schleife dran* — *3 Punkte*

Jetzt wollen wir die Weihnachtsgrüße noch schön gestalten. Dazu schreiben wir ein Funktion `fbox` (angelehnt an das LaTeX-Kommando gleichen Namens), welche einen String aus Zeilen (wie er von `banner` zurückgegeben wird) einrahmt:

```
fbox :: Char → String → String
putStr (fbox '@' "foo\nbar\nbaz") ⇝
@@@@@@@
@ foo @
@ bar @
@ baz @
@@@@@@@
```

Wenn die Zeilen im zweiten Argument von `fbox` nicht gleich lang sind soll `fbox` ein Fehler zurückgeben (`error`). Schreiben Sie dazu eine Funktion, welche prüft ob in einer Liste von Listen alle die gleiche Länge haben: (Was ist der einzig sinnvolle Wert für `check []`?)

```
check :: [[α]] → Bool
check ["abc", "def", "ghi"] ⇝ True
check [[0,1,2], [3, 4], [5,6,7,8]] ⇝ False
```

Falls die Zeilen nicht die gleiche Länge haben, können wir sie auf die gleiche Länge bringen. Dazu schreiben wir eine Funktion, welche eine Liste von Listen auf die gleiche Länge (der längsten Liste) bringt, indem sie ein gegebenes Zeichen entsprechend oft hinten anfügt:

```
eq_length :: α → [[α]] → [[α]]
eq_length 0 [[1,2], [3,4,5], [6]] ⇝ [[1,2,0],[3,4,5],[6,0,0]]
```

Hier benötigen wir eine Spezialisierung auf Zeichenketten, in der mit dem Leerzeichen aufgefüllt wird:

```
eq_lines :: String → String
eq_lines "ab\ncdef\ng" ⇝ "ab␣␣\ncdef\ng␣␣␣\n"
```

Jetzt können wir die beiden Aufgabenteile zusammenfügen und einen schönen Weihnachtsgruß erzeugen:[1]

```
putStr (FBox.fbox '=' (banner '#' "Season's␣Greetings!")) ⇝
============================================================================================================================================================
=   ###                                               #                     ###                                                                      #     =
=  #   #                                              #                    #   #                            #       #                                #     =
=   #      ####    ####    ####    ####    # ##      #     ####           #       ## ##    ####    ####     #              # ##    #####   ####      #     =
=    #    #    #       #  #       #    #   ##  #          #               #        ##  #  #    #  #    #  #####    ##      ##  #  #    #  #          #     =
=     #   ######   #####   ####   #    #   #   #           ####           #   ###  #      ######  ######    #       #      #   #  #    #   ####      #     =
=  #   #  #       #    #       #  #    #   #   #               #           #   ##  #      #       #         #       #      #   #   #####       #           =
=   ###    ####    #### # #####    ####    #   #          #####             ### # ##       ####    ####      ##   #####    #   #       #  #####      #     =
=                                                                                                                                  ####                    =
============================================================================================================================================================
```

[1] Die Bitmap für die großen Zeichen stammt aus dem Internet und enthält natürlich keine Umlaute— daher bleiben wir erstmal englisch.