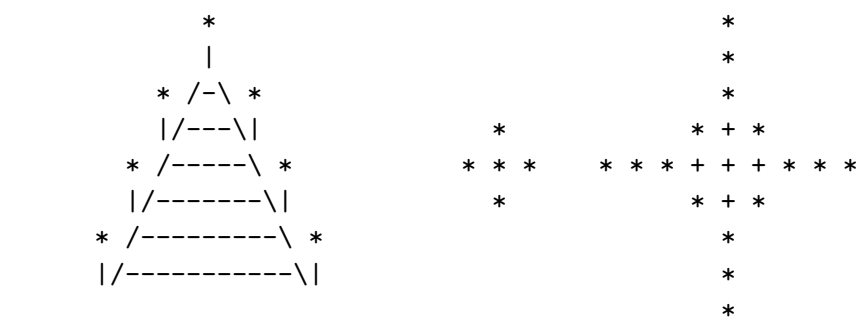
Zusatzübung **Informatik A** **WS 11/12**

Klaus Kriegel **Abgabe: 08.01.2014, 10:00 Uhr**



Die Bearbeitung dieser Weihnachtsübung ist freiwillig und alle dabei erzielten Punkte werden als Bonuspunkte gewertet. Auf den ersten Blick sieht es nach einer Spielerei aus, aber man kann dabei sehr viel über höhere Listenfunktionen lernen.

Die Funktion `putStr` wird später in der Vorlesung noch genauer behandelt. Sie müssen für diese Übung nur wissen, dass sie als Eingabe einen String erhält und diesen dann auf dem Bildschirm darstellt, wobei das Newline-Symbol `\n` die Darstellung in die nächste Zeile überleitet. Außerdem muss man beachten, dass man zur Darstellung des \ ein \\ angeben muss. Die Spitze des links oben dargestellten Weihnachtsbaums erhält man also durch:

```
putStr "         *  \n         | \n      * /-\\ * \n      |/---\\| \n"
```

Dabei müssen natürlich auch die linksstehenden Leerzeichen genau abgezält werden.

Die Aufgabe besteht darin, die Zeichnungen der rechts dargestellten Weihnachtssterne automatisch zu generieren. Allgemein soll die obere Spitze eines solchen Sterns durch ein 1×3-Rechteck gebildet werden, das mittig auf einem 3×3-Rechteck steht, welches wieder mittig auf einem 5×3-Rechteck steht usw. Die Basisrechtecke der 4 Spitzen werden an ihren Mittelpunkten verschmolzen. Der n-te Stern füllt somit ein $3\cdot(2n+1)\times3\cdot(2n+1)$-Gitter. Alle Randpunkte der Sterne sollen mit `*`-Symbolen und alle inneren Punkte mit `+`-Symbolen gezeichnet werden. Oben sieht man den nullten und ersten Stern aus der Serie, umseitig ist der dritte Stern dargestellt. Am Ende der Aufgabe sollte eine Funktion `stern :: Int -> [Char]` entstehen, die bei Eingabe n den String des n-ten Sterns konstruiert. Sie sollten sich dabei möglichst an die folgenden Instruktionen halten, deren Einzelschritte am Beispiel von `n=3` in der Abbildung illustriert sind.

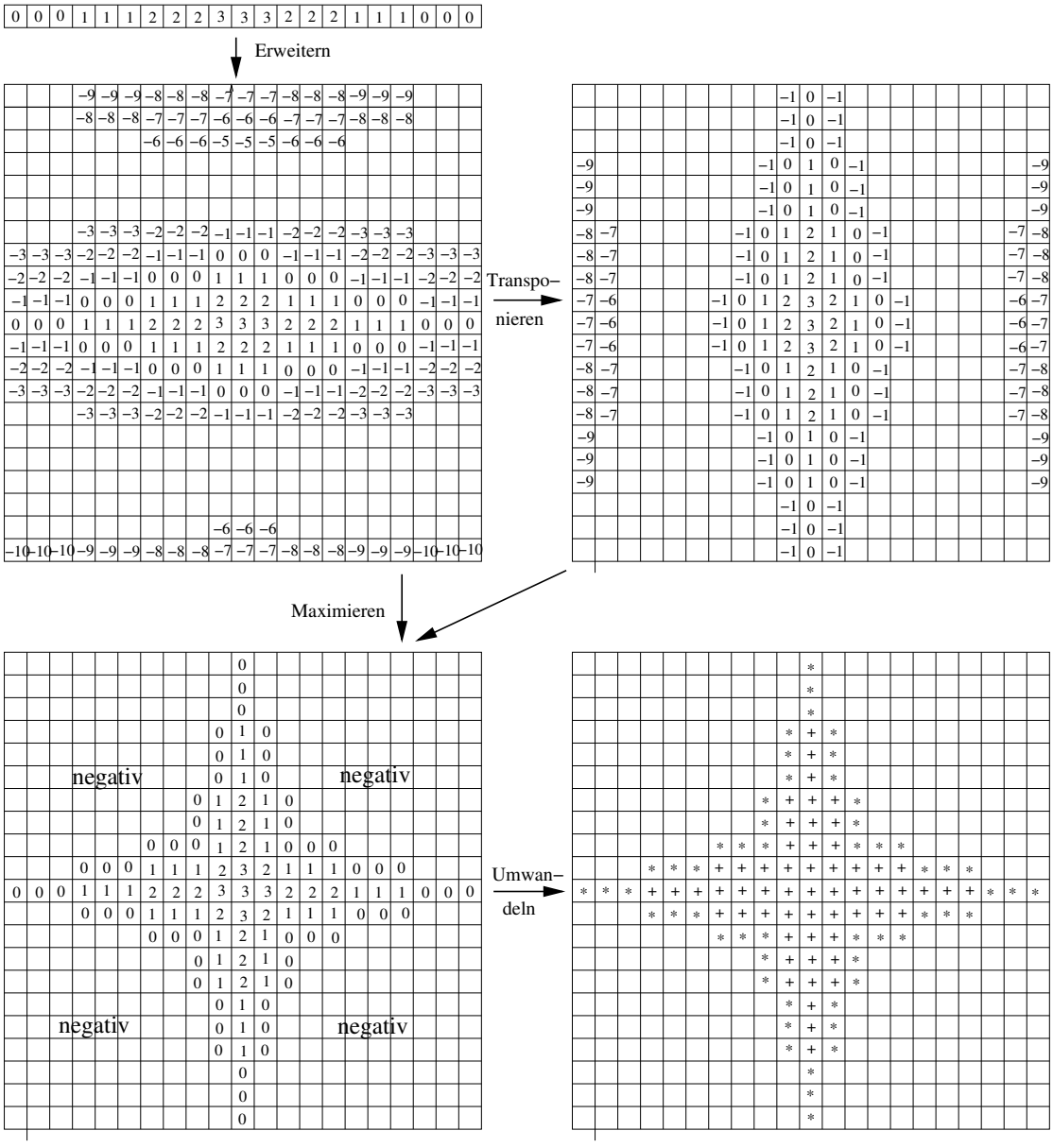
a) Zuerst wird eine Funktion `initList :: Int -> [Int]` konstruiert, um bei Eingabe `n` die Liste `[0,0,0,1,1,1,...,n,n,n,...,1,1,1,0,0,0]` zu erzeugen.

b) Im nächsten Schritt denkt man sich die initiale Liste als mittleres Element einer Liste von Listen und erweitert diese wiederholt nach beiden Seiten durch Verkleinern aller Elemente um 1, bis man ein quadritisches Format erreicht hat. Das kann durch eine Funktion `extend :: Int -> [[Int]] -> [[Int]]` realisiert werden, wobei der erste

Parameter die Anzahl der nach oben/unten noch fehlenden Zeilen zählt. Sie sollten hier möglichst die Funktion `map` verwenden.

c) In dem bisherigen Schema repräsentieren die nichtnegativen Werte die beiden horizontalen Strahlen des Sterns. Implementieren Sie nun `transpose :: [[a]] -> [[a]]` um ein solches Schema um 90° zu drehen, also die (gedachten) Spalten in Zeilen umzuwandeln. Hier sollte man eine Faltung einbauen (siehe Hinweis)!

d) Nun kann man die Funktion `maxListOfLists` aus der 9. Übung verwenden, um das Zahlenschema links unten zu erzeugen. Im Ergebnis sollte jetzt eine Funktion `SternNum :: Int -> [[Int]]` entstanden sein.



e) Definieren Sie eine Funktion `convert :: Int -> Char` um positive Zahlen in `+`,

Nullen in `*` und negative Zahlen in Leerzeichen umzuwandeln und wenden Sie diese mit `map` auf alle Elemente des Zahlenschemas an. Außerdem muss jede Zeile noch ein Newline-Symbol erhalten. Jetzt haben wir eine Funktion
`SternChar :: Int -> [[Char]]`.

f) Verwenden Sie eine Faltung, um die Zeilen zu einem String zusammenzuziehen.

g) Wenn Sie diesen String mit der Funktion `putStr` ausgeben,werden Sie einen horizontal gestauchten Stern erkennen, weil die Zeilenabstände größer als die Zeichenabstände in einer Zeile sind. Korrigieren Sie diesen Schönheitsfehler durch eine Funktion `widening :: [Char] -> [Char]`, die zwischen zwei Zeichen jeweils ein Leerzeichen einfügt.

Weitere Hinweise:

1. Punkteverteilung: $1 + 2 + 3 + 1 + 3 + 1 + 1$. Die Punkte werden zur 10. Übung gutgeschrieben.

2. Wenn Sie einzelne Teilaufgaben nicht realisieren können, kann man Zwischenergebnisse auch ausschreiben (für $n = 1$ oder $n = 2$) und damit weiterrechnen.

3. Teil c) könnte die meisten Sorgen bereiten. Hier ist ein Vorschlag, wie man es angehen kann:
   Man definiert zuerst eine Funktion `appendLeft :: [a] -> [[a]] -> [[a]]`, um die Elemente der ersten Liste jeweils als Anfangselemente der Listen aus der Listenliste anzuhängen. So sollte `appendLeft [1,2,3,4,5] [[6,7],[],[8,9]]` das Ergebnis `[1,6,7],[2],[3,8,9],[4],[5]` erzeugen. Die Funktion `appendLeft` kann dann sehr schön mit einer Faltung kombiniert werden.
   Außerdem kann man `appendLeft` auch gut in e) zum Hinzufügen der Newline-Symbole verwenden (in diesem Fall kombiniert mit der vordefinierten Listenfunktion `replicate :: Int -> a -> [a]` zur Erzeugung einer Liste mit der `k`-fachen Wiederholung eines Elements).