Übungen zu Softwareentwicklung III, Funktionale Programmierung

Blatt 6, Woche 7

Leonie Dreschler-Fischer

WS 2015/2016

Ausgabe: Freitag, 20.11.2015,

Abgabe der Lösungen: bis Montag, 30.11.2015, 12:00 Uhr per email bei den Übungsgruppenleitern.

Ziel: Rekursion: Die Aufgaben auf diesem Zettel dienen dazu, sich mit dem Entwurf von rekursiven Funktionen vertraut zu machen. Sie entwerfen linear rekursive Funktionen und üben die unterschiedlichen Formen von Rekursion zu unterscheiden.

Bearbeitungsdauer: Die Bearbeitung sollte insgesamt nicht länger als 5 Stunden dauern.

Homepage:
http://kogs-www.informatik.uni-hamburg.de/~dreschle/teaching/Uebungen_Se_III/Uebungen_Se_III.html

Bitte denken Sie daran, auf den von Ihnen eingereichten Lösungsvorschlägen *Ihren Namen und die Matrikelnummer, den Namen der Übungsgruppenleiterin / des Übungsgruppenleiters* und *Wochentag und Uhrzeit der Übungsgruppe* anzugeben, damit wir ihre Ausarbeitungen eindeutig zuordnen können.

1 Formen der Rekursion

(Bearbeitungszeit 1/2 Std.), 10 Pnkt.

Gegeben seien die folgenden Funktionsdefinitionen:

```
(define (kopfstueck n xs)
 ;; das Kopfstueck einer Liste: die ersten n Elemente
 (cond
  ((null? xs) '())
  ((= 0 n) '())
  (else (cons (car xs)
              (kopfstueck (- n 1) (cdr xs))))))

(define (endstueck n xs)
 ;; das Endstueck einer Liste entfernen
 (cond
  ((null? xs) '())
  ((= 0 n) xs)
  (else (endstueck (- n 1) (cdr xs)))))

(define (merge rel<? xs ys)
 ;; mische zwei vorsortierte Listen xs und ys
 ;; entsprechend der Ordnungsrelation rel<?
 (cond ((null? xs) ys)
   ((null? ys) xs)
    (else
      (let ((cx (car xs))
            (cy (car ys)))
           (if (rel<? cx cy)
             (cons cx (merge rel<? (cdr xs) ys))
             (cons cy (merge rel<? (cdr ys) xs)))))))

(define (merge-sort rel<? xs)
 ;; Sortiere eine Liste xs entsprechend
 ;; der Ordnungsrelation rel<?
  (let ((n (length xs)))
    (if (<= n 1) xs
      (let*((n/2 (quotient n 2))
            (part1 (kopfstueck n/2 xs))
            (part2 (endstueck n/2 xs)))
           (merge
              rel<?
              (merge-sort rel<? part1)
              (merge-sort rel<? part2))))))
```

Welcher Typ von Rekursion liegt jeweils vor?
(Mehrfachnennungen sind möglich)
Begründen Sie Ihre Entscheidung.

	lineare Rekursion	Baum-Rekursion	geschachtelte Rekursion	direkte Rekursion	indirekte rekursion
`kopfstueck`	*ja/nein*	*ja/nein*	*ja/nein*	*ja/nein*	*ja/nein*
`endstueck`	*ja/nein*	*ja/nein*	*ja/nein*	*ja/nein*	*ja/nein*
`merge`	*ja/nein*	*ja/nein*	*ja/nein*	*ja/nein*	*ja/nein*
`merge-sort`	*ja/nein*	*ja/nein*	*ja/nein*	*ja/nein*	*ja/nein*

2 Rekursives Sortieren von Listen

(Bearbeitungszeit 1 1/2 Std.)

Der merge-sort Algorithmus aus Aufgabe 1 ist ein Beispiel für einen rekursiven Sortieralgorithmus. Im Folgenden wollen wir uns mit einigen Sortieralgorithmen befassen. Imperativ implementierte Varianten dieser Verfahren sind Ihnen möglicherweise aus den Veranstaltungen Softwareentwicklung I und Softwareentwicklung II bekannt. *Insertion-Sort* und *Quicksort* sind Standardverfahren, die Sie bspw. im Cormen beschrieben finden (1).

2.1 Insertion-Sort

5 Pnkt.

Implementieren Sie den *Insertion-Sort*-Algorithmus mittels linearer rekursion (gerne auch Endrekursion). *Insertion-sort* ist ein „in-place"-Algorithmus, der folgendermaßen abläuft: Wähle ein beliebiges Element einer unsortierten Liste und füge es an der richtigen Stelle in der Liste ein. Wähle anschließend das nächste Element aus. Wende dieses Verfahren wiederholt an, bis alle Elemente einsortiert sind.

2.2 Quicksort

7 Zusatzpnkt.

Implementieren Sie den Quicksort-Algorithmus mit einer Baumrekursion. Der Quicksort Algorithmus ist ein klassicher Vertreter der sogenannten „divide & conquer"-Verfahren. Eine unsortierte Liste wird in zwei Hälften geteilt. Dazu wählt man beliebig ein Pivot-Element. Alle Elemente linksseitig des Pivot-Elements und alle Elemente rechtsseitig des Pivot-Elements werden wiederum von der Quicksort-Funktion sortiert.

2.3 Testen der Sortierverfahren mit einer Liste von Bildern

3 Pnkt.

Jetzt sollen die Sortierverfahen getestet werden. Da eine einfache Liste von Zahlen langweilig ist, werden Bilder sortiert. Erweitern Sie Ihre Sortierfunktionen, sodass die Bilder nach verschiedenen Kriterien sortiert werden können (z.B. Breite, Höhe, mittlere Helligkeit, usw.).

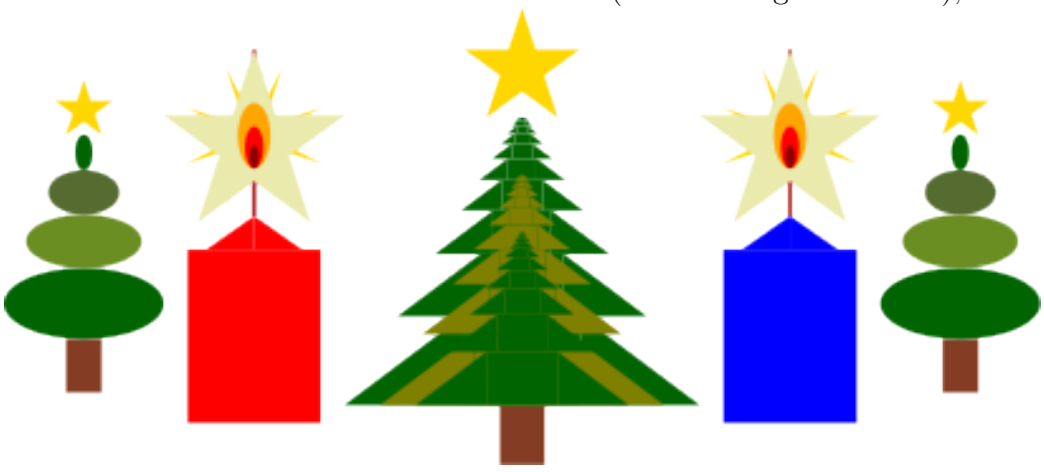
Laden Sie das Modul mit `(require 2htdp/image)` Erzeugen Sie eine Liste von Bildern, etwa so:

```
(define icons
  (list
   (star-polygon 35 5 2 "solid" "gold")
   (ellipse 44 44 "solid" "red")
   (rectangle 38 38 "solid" "blue")
   (isosceles-triangle 45 65 "solid" "darkgreen")))
```

Tipp: Wenn Sie die Vergleichsfunktion als Argument der Sortierfunktion angeben, können Sie nach beliebigen Kriterien sortieren.

3 Ihre Nikolausaufgabe

(Bearbeitungszeit 3 Std.), 20 Pnkt.



Machen Sie sich mit dem Racket-Modul `"2htdp/image"` vertraut (siehe DrRacket-Hilfezentrum im Help-Menü).

In dieser Aufgabe ist Ihre Kreativität gefragt! Verwenden Sie grafische Elemente, wie Kreis, Rechteck usw., um ein festliches, weihnachtliches Bild zu komponieren, beispielsweise mit einem geschmückten Tannenbaum, Kerzen, Stapeln von Geschenken usw. Das Bild soll wiederholte Elemente enthalten, die rekursiv zu erzeugen sind. Insbesondere lassen sich ansehnliche Verzweigungsstrukturen durch Baumrekursion erzeugen. Das Programm soll modular aufgebaut sein und gut kommentiert werden.

Ein Beispiel: Die kleinen Bäumchen rechts und links wurden aus olivgrünen Ellipsen, einem gelben Stern und einem braunen Rechteck zusammengesetzt, alles mit `above/align` zentriert übereinandergestapelt:

```
(define baum1 (above/align
               "center"
               ;; der Stern an der Spitze
               (star-polygon 40 5 2 "solid" "gold")
               ;; die Zweige
               (ellipse 20 40 "solid" "darkgreen")
               (ellipse 80 50 "solid" "darkolivegreen")
               (ellipse 130 60 "solid" "olivedrab")
               (ellipse 180 80 "solid" "darkgreen")
               ;; der Stamm
               (rectangle 40 60 "solid" "brown")
               ))
```

Erreichbare Punkte: 38

Erreichbare Zusatzunkte: 7

Literatur

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms (3. ed.)*. MIT Press, 2009.