Grundkurs Common-Lisp 6. Übung (25. Nov 1999)

(Destruktive) Rekursion und Funktionsobjekte

1 Schon wieder Rekursion (2 + 3 = 5 Punkte)

Schreibe eine Funktion `for-all!` die eine Funktion `f` und eine Liste `l` nimmt, und `f` an alle elemente in `l` appliziert.

```
(for-all #'print '(f o o))
F
O
O
-> (F O O)
```

Erweitere die Funktion so, dass eine Liste mit den Ergebnissen der Funktionsapplikation zurückgegeben wird

```
(for-all2 #'1+ '(1 2 3))
-> (2 3 4)
(for-all2 #'list '(1 2 3))
-> ((1) (2) (3))
```

2 Schon wieder mehr Rekursion (3 + 4 = 7 Punkte)

Schreibe eine Funktion, die eine Funktion und eine beliebige Liste nimmmt und die Funktion an alle Atome appliziert.
Beispiele:

```
(for-leaves #'print '(1 (2 . 3) ((((42))))))
1
2
3
```

Erweitere Die Funktion so dass eine *isomorphe* Liste mit den Ergebnissen der Funktionsapplikationen zurückgegeben wird.
Beispiel:

```
(for-leaves2 #'1+ '(1 (2 . 3) ((((42))))))
-> (2 (3 . 4) ((((43))))))
(let ((s 0))
  (for-leaves2 #'(lambda (x)
                   (cond ((numberp x)
                          (setf s (+ x s))
                          s)
                         (t x)))
               '(1 ((f o . 4) 5) (((16 . hej))) slut)))
-> (1 ((F O . 5) 10) (((26 . HEJ))) SLUT)
```

3 Destruktive Rekursion (3 + 4 = 7 Punkte)

Erweitere die Funktion aus der ersten Aufgabe so daß das Ergebnisse die jeweilige Element ersetzen.
Beispiel

```
(setf l '(1 2 3))
(for-all! #'1+ l)
-> OK
l
-> (2 3 4)
```

4 Destruktive Restrekursion 2 rekursive Schmökpunkte

Anbei eine mögliche restrekursive Definition von `reverse`. Die Schmökaufgabe besteht diesmal darin, sie *destruktiv* umzuschreiben. Wichtig ist zu zeigen, dass die erweiterte Funktion in der Tat destruktiv ist, und dass keine extra cons-Zellen verbraucht werden.

```
(defun my-reverse (l)
  (r-h (rest l) (list (first l))))
(defun my-reverse-help (l res)
  (cond ((endp l) res)
        (t (my-reverse-help (rest l) (cons (first l) res)))))
```

Hinwies: Um den Speichervebrauch anzuzeigen gibt es in CommonLisp die vordefinierte Funktion `time`, die angibt wieviel Speicher (Cons bzw andere Bytes) und Zeit ein Funktionsaufruf in Anspruch nimmt. Vergiss dabei nicht, deine Funktionen zu kompilieren – `compile` heißt die Funktion dafür.

Abgabetermin ist der 2. Dezember 1999