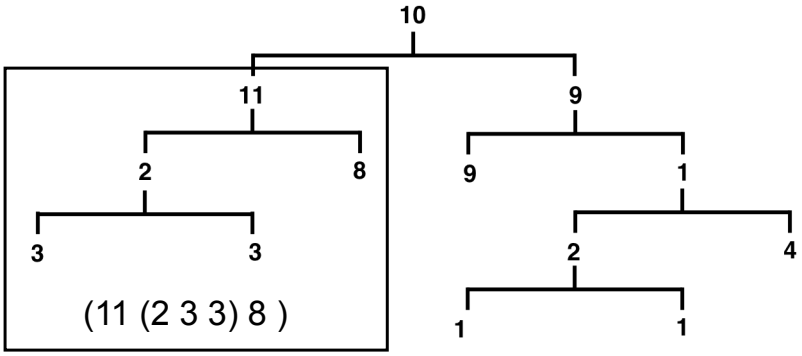
Übungsblatt 3

Ausgabe: 17.12.2018
Abgabe: 7.1.2019

Aufgabe 1: Mobiles

Ein Mobile ist eine Art abstrakter Skulptur, die aus Teilen besteht, die sich relativ zueinander bewegen können. Normalerweise besteht es aus Objekten, die mit feinen Schnüren an horizontalen Stöckchen befestigt und frei beweglich aufgehängt sind.



Wir können ein bestimmtes einfaches Mobile rekursiv definieren als aufgehängtes Objekt oder als Stöckchen mit daran hängenden (Teil-) Mobiles an jedem Ende. Wenn wir unterstellen, dass jedes Stöckchen in seiner Mitte aufgehängt ist, können wir das Mobile als binären Baum auffassen. Einzelne aufgehängte Objekte sollen durch Zahlen, entsprechend dem Gewicht der Objekte dargestellt werden. Dabei entspricht das erste Element dem Gewicht des Stöckchens, die beiden anderen Elemente stellen die Teil-Mobiles dar. Somit lässt sich z.B. der unterste Zweig dieses Mobiles durch die dreielementige Liste (2 1 1) darstellen.
Kompliziertere Teil-Mobiles werden durch geschachtelte 3-Element-Listen dargestellt. Im Beispiel das 2-stufige Teil-Mobile rechts durch die geschachtelte Listenstruktur (1 (2 1 1) 4)
Das abgebildete Mobile entspricht dann der Liste (10 (11 (2 3 3) 8) (9 9 (1 (2 1 1) 4)))
Ein Mobile sollte im Gleichgewicht sein. Das bedeutet, dass 2 Mobiles, die an den gegenüberliegenden Enden eines Stöckchens hängen, das gleiche Gewicht haben müssen. Das ist bei dem abgebildeten Mobile der Fall. Schreiben Sie eine Funktion `mobilep`, die für ein übergebenes Mobile feststellt, ob es das Mobile im Gleichgewicht ist. Der Rückgabewert ist `NIL`, bei Ungleichgewicht, im andern Fall das Gewicht.
`(mobilep '(10 (11 (2 3 3) 8) (9 9 (1 (2 1 1) 4))))` → `64`
`(mobilep '(10 (11 (2 3 3) 8) (9 9 (1 (2 1 2) 4))))` → `NIL`

Achten Sie bei der Implementierung darauf, dass der Code möglichst kompakt und gut verständlich ist (dass er korrekt ist, versteht sich von selbst); Effizienz ist zweitrangig. Nutzen Sie Rekursion!.

Aufgabe 2: Methoden zur Listenmanipulation

(a) **Zahlen aus Liste entfernen**: Schreiben Sie eine Funktion `liste-ohne-zahlen` um aus einer Liste alle Zahlen zu entfernen.

`(liste-ohne-zahlen '(1 a b 2 c 4 5))` → `a b c`

Freiwillig: `liste-ohne-zahlen-R` - entfernt auch aus Unterlisten die Zahlen.

(b) **Funktionsdefinition zusammenbauen**: Schreiben Sie eine Funktion `funkdef-1arg` zum Zusammenbauen einer Funktion mit einem Parameter.
Parameter:
fktname: ein Symbol
op: der Name einer Lisp-Funktion
const, var: Symbole
Wert: Eine Funktionsdefinition für `fktname` mit dem Parameter `var`, sodass `op` auf `const` und `var` angewendet werden.
Aufrufbeispiele:
`(funkdef-1arg 'add1 '+ 1 'zahl)` → `(defun add1 (zahl) (+ zahl 1))`
`(funkdef-1arg 'add2 '+ 2 'zahl)` → `(defun add2 (zahl) (+ zahl 2))`
`(funkdef-1arg 'mult3 '* 3 'zahl)` → `(defun mult3 (zahl) (* zahl 3))`

(c) **Funktionsdefinition zusammenbauen und Auswerten**: Schreiben Sie eine Funktion `funkdef-1arg+eval` zum Zusammenbauen einer Funktion mit einem Parameter.
Parameter:
fktname: ein Symbol
op: der Name einer Lisp-Funktion
const, var: Symbole
arg: eine Zahl
Effekt: Durch Aufruf der Funktion funkdef-1arg soll eine entsprechende Funktionsdefinition erzeugt und ausgewertet werden. Anschließend ist die neu definierte Funktion mit arg anzuwenden.
Wert: Der Wert, der bei Anwendung von fktname auf arg entsteht.
Aufrufbeispiele
`(funkdef-1arg+eval 'add2 '+ 2 'zahl 55)` → `57`
Seiteneffekt: Es wird eine Funktionsdefinition für add2 gebildet und ausgewertet. add2 wird dann mit dem Argument 55 aufgerufen werden: `(funkdef-1arg+eval 'add2 '+ 2 'zahl 55)`
`(funkdef-1arg+eval 'sub5 '- 5 'zahl 55)` → `50`
Seiteneffekt: Definition der Funktion `sub5`.

(d) **Listen-Operation anwenden**: Schreiben Sie eine Funktion `listenop`, welche die als Parameter übergebene Operation auf zwei, ebenfalls als Parameter übergebene Listen anwendet.
Geben Sie 2 Lösungen ab: eine mit und eine ohne Verwendung von `funcall`.

`(listenop 'list '(3 4) '((c d) (e f)))` → `((3 4) ((c d) (e f)))`
`(listenop 'append '(3 4) '((c d) (e f)))` → `(3 4 (c d) (e f))`

(e) **Unterlisten erstellen**: Schreiben Sie eine Funktion `key-unterliste`, die eine Liste mit Unterlisten untersucht. Dabei wird die Unterliste als Ergebnis zurückgegeben, die einen übergebenen Ausdruck als erstes Element enthält.

`(key-unterliste 'c '((a 4) (b 9) (c d) (e f)))` → `(c d)`

Die Aufgabe soll mit `member` und einer geeigneten Testfunktion (wird der Funktion `member` als Schlüsselwortparameter übergeben) gelöst werden.