5. Übung ALP 1: Funktionale Programmierung WS 14/15

Klaus Kriegel **Abgabe: 26.11.2014, 10:15 Uhr**

Aufgabe 1: nichtprimitive Rekursion (1 + 2 + 4 Punkte)

Wir erweitern die rekursive Definition der Fibonaccizahlen um einen zusätzlichen Summanden mit $a_0 = a_1 = 0$ und $a_2 = 1$ als Rekursionsverankerung und der Regel $a_n = a_{n-1} + a_{n-2} + a_{n-3}$ für alle $n \geq 3$.
a) Implementieren Sie die Berechnung von a_n als Funktion `superFib ::Int -> Int` mit Fallunterscheidung (Guards).

b) Bezeichne b_n die Gesamtanzahl der Funktionsaufrufe von `superFib` bei der Berechnung von `superFib n`. Implementieren Sie eine Funktion `countCalls ::Int -> Int` zur Bestimmung von b_n. Erstellen Sie die Liste der Werte b_n für alle $1 \leq n \leq 20$ und überprüfen Sie daran die Gültigkeit der Ungleichung $b_n \leq 2^n - 1$.

c) Beweisen Sie $a_n \leq 2^{n-2}$ für alle $n \in \mathbb{N}$ und $b_n \leq 2^n - 1$ für alle $n \in \mathbb{N}^+$ mit vollständiger Induktion.

Aufgabe 2: Tupel (2 + 2 + 2 Punkte)

Wir führen den Datentyp `type Zeit = (Int,Int)` zur Dartellung einer Uhrzeit als Paar (x, y) mit $x \in \{0, 1, \ldots, 23\},\ y \in \{0, 1, \ldots, 59\}$ als Stunden/Minutenwert.

a) Implementieren Sie Funktionen `addStd, addMin ::Zeit -> Int -> Zeit`, um die Uhr um eine bestimmte Anzahl von Stunden oder Minuten weiterzustellen. Sie können vorausetzen, dass der übergebene `Int`-Parameter nicht negativ ist, müssen aber die Sprünge über die Stunden- oder Tagesgrenzen beachten (`addMin (23,58) 4` muss das Ergebnis `(0,2)` haben).

b) Implementieren Sie eine Funktion `dauer ::Zeit -> Zeit -> Zeit`, die für zwei gegebene Zeiten `t1 t2` die Zeitdauer berechnet, wann zum ersten Mal nach der Zeit `t1` die Zeit `t2` angezeigt wird (bei Gleichheit sollte das Ergebnis `(0,0)` erzeugt werden).

c) Implementieren Sie eine Funktion `anzeige ::Zeit -> [Char]` mit der eine Zeit `(x,y)` in einen String der Form ‘‘`x:y Uhr`’’ umgewandelt wird, wobei der Minutenwert immer zweistellig dargestellt werden soll.

Aufgabe 3: Strings (4 Punkte)

Implementieren Sie `deleteABs, doubleDigits, shiftDigits ::[Char] -> [Char]` mit folgenden Eigenschaften:
`deleteABs` streicht aus einem String die Zeichen `'A'` und `'B'`, `doubleDigits` verdoppelt jede im String vorkommende Ziffer und `shiftDigits` verschiebt jede im String vorkommende Ziffer auf die zyklisch nächste Ziffer.

Aufgabe 4: Listenrekursion (1 + 1 + 2 Punkte)

Implementieren Sie die folgenden Funktionen elementar mit Listenrekursion (d.h. keine

vordefinierten Funktionen verwenden!):

a) `countPos ::[Int] -> Int` zählt die Anzahl der positiven Einträge in der Liste

b) `monIncrPrefix ::[Int] -> [Int]` gibt den Anfangsteil (Präfix) der Eingabeliste aus, der schwach monoton wachsend ist (`monIncrPrefix [2,3,3,1,4]` $\rightsquigarrow$ `[2,3,3]`)

c) `parSum ::[Int] -> [Int]` berechnet die Partialsummenfolge (`parSum [2,3,0,-1,4]` $\rightsquigarrow$ `[2,5,5,4,8]`)

Hinweis: Nutzen Sie eine geeignete Hilfsfunktion `help ::(Int,[Int]) -> (Int,[Int])`.