

Algorithmen und Programmierung I

WS 2004 / 2005

Übung 7 (keine Punkte)

Ausgabe: Mo. 6.12.2004

Besprechung offener Fragen: Mi. 15.12.2004

Die Übung dient der Vorbereitung der Klausur. Sie wird nicht bewertet. Sie bereiten sich am Besten darauf vor, wenn Sie versuchen, die Aufgaben in einem Zeitrahmen von ca. 2 Stunden zu lösen, ohne dabei auf Material wie Folien, Skript, Bücher zurückzugreifen. Danach versuchen Sie mit Hilfe von Literatur die Aufgaben zu lösen, die Sie nicht geschafft haben bzw. um die Lösungen zu überprüfen.

Damit stellen Sie sich auf die Klausurbedingungen ein: es dürfen keine Unterlagen benutzt werden. Ferner können Sie damit Ihre Klausurstrategie trainieren: Es empfiehlt sich zum Beispiel, Aufgaben, für die man nach 5 Minuten noch keine Lösungsidee hat, erst mal links liegen zu lassen.

Die Aufgaben sind vom Schwierigkeitsgrad ähnlich wie die in der Klausur. Es wird sich aber nicht um kleine Varianten (z.B. `foldr` statt `foldl`) handeln, sondern um gänzlich andere Aufgaben.

Mit Unterlagen erscheint manche Aufgabe eventuell ganz leicht, weil z.B. die Definition in Büchern steht, in der VL oder Tutorium besprochen wurde oder beides. Denken Sie aber immer daran, dass Sie auf diese Hilfsmittel nicht zurückgreifen dürfen. Da könnte schon die Definition von `foldr1` zur Hürde werden....

Ganz wichtig: wer in der Klausur mogelt, fällt auf der Stelle durch. Stellen wir im Zuge der Korrekturen fest, dass Klausurteilnehmer offensichtlich voneinander abgeschrieben haben, fallen beide durch.

Aufgabe 7.1

a) Die Funktion `foldl1` hat die Signatur

`foldl1 :: (a->a->a) -> a -> [a] -> a`

Sie ist damit spezieller als die Signatur von `foldl`.

Warum? (Max 2 Sätze Begründung)

b) Drücken Sie `((map f) . (filter p)) xs` durch die ZF-Notation aus.

c) Welchen Typ hat die Funktion `zweimal f = f . f` ?

d) Verallgemeinern Sie `zweimal` zu einer Funktion `iteriere`, die `f` `n` mal ausführt, `n >= 1`.

Aufgabe 7.2

a) Schreiben Sie eine Funktion `removeEven`, die aus Listen von ganzen Zahlen so lange Elemente vom Anfang entfernt, bis man auf ein ungerade Zahl stößt. Die Restliste soll Wert der Funktion sein.

Beispiel: `removeEven [2,-4,54,3,17,8,2,5] = [3,17,8,2,5]`

b) Abstrahieren Sie die Lösung aus a) zu einer Funktion `dropWhile`, die alle Anfangselemente einer Liste entfernt, die eine Eigenschaft `p` besitzen.

c) definieren Sie `removeEven` mit Hilfe von `dropWhile`.

Aufgabe 7.3

Gegeben sind zwei Implementierungen `f, f' :: Int -> Int` eines Problems. Ein Beispiel wäre die naive und die endrekursive Implementierung der Fakultätsfunktion.

Gesucht ist eine Funktion `test`, die Funktionswerte von `f` bzw. `f'` für die Werte `[n..m]` berechnet und `True` als Wert liefert, wenn die Werte paarweise gleich sind, `False` sonst.

Aufgabe 7.4

a) Gegeben die Definition `g str = [c | c <- str, c == "x"]`
Welche Ausgabe liefert `g "xerox Kopien aus Xanten"`

`"xx"`
`"c"`
`"xxx"`
`"xxX"`
`"xerox Xanten"`
keine der obigen Alternativen

b) `(x:xs)` ist äquivalent zu

`x ++ xs`
`[x] ++ xs`
`[x] ++ [xs]`
allen drei Ausdrücken
keinem

c) Sind die folgenden Ausdrücke und Definitionen typkorrekt, wenn nein: Begründung, wenn ja, welchen Typ haben sie?

- 1) `[["Abel"], ["Bebel"]] [Cebel]`
- 2) `double = map f`
`where f :: Char -> String`
`f c = "cc"`
- 3) `concat double "abc" -- double aus 2)`
- 4) `concat (double "abc")`
- 5) `(concat . double) "abc"`

Aufgabe 7.5

a) Definieren Sie eine endrekursive Funktion `insertBetween`, die zwischen je zwei Elementen und am Ende einer nichtleeren Liste einen Wert einfügt. Liste und Wert sind Argumente der Funktion.

Beispiel: `insertBetween [1,3,4,6] 2 = [1,2,3,2,4,2,6,2]`

b) Definieren Sie mit Hilfe der ZF-Notation eine Funktion `duplicate`, die jedes Element der Argumentliste dupliziert.

Beispiel: `duplicate [1,2,3] = [1,1,2,2,3,3]`

Aufgabe 7.6

a) Die Funktion `substituteW` soll aus einer Wortliste jedes Vorkommen eines missliebigen Wortes durch so viele 'x' ersetzen, wie das Wort lang ist. Welche Alternative(n) (1) bis (4) ist / sind korrekt, welche sind falsch (knapp begründen!)?

```

replaceW badWord word
  | badWord == word = rep (length badWord) 'x'
  | otherwise       = word
  where rep n x = [ x | k <- [1..n]]

```

```

substituteW :: String -> [String] -> [String]
(1) substituteW badWord = map (replaceW badWord)
(2) substituteW badWord = map . replaceW badWord
(3) substituteW badWord = replaceW badWord . map
(4) substituteW badWord = map badWord . replaceW

```

b) Verallgemeinern Sie `substituteW` zur Funktion `multiSubW`, die jedes Wort aus einer Liste von missliebigen Wörtern in der Wortliste wie oben beschrieben ersetzt.

Aufgabe 7.7

Eine Adresse bestehe aus einer Titelzeile ("Herr" oder "Frau"), dem Vornamen und dem Name eines Adressaten, einer optionalen Straßen- oder Postfachangabe einschließlich Haus- oder Postfachnummer sowie einer Ortsangabe, die die Postleitzahl enthält. Jede dieser Angaben steht in einer separaten Zeile (letztes Zeichen der Zeichenkette '\n'). Definieren Sie die Syntax in BNF und mit Hilfe von Syntaxdiagrammen.

Aufgabe 7.8

a) Erläutern Sie knapp:

- den Unterschied zwischen einem Übersetzer und einem Interpretierer
-
- was ein deterministischer Algorithmus ist
-
- was man unter der Eigenschaft Robustheit eines Algorithmus versteht.
-
- was man unter einer partiellen, surjektiven Funktion $f : A \rightarrow B$ versteht
-
- was ein assoziativer Operator ist

b) Geben Sie den Wert der korrekten Haskell-Ausdrücke an. Inkorrekte sollen ohne Begründung angekreuzt werden.

```

foldr (/) 1 [1,2,3]
foldl (/) 1 [1,2,3]
take 2 . reverse [1,2,3,4]
(take 2 . reverse) [1,2,3,4]
zip [x | x <- [1..5], odd x] [x | x <- [1..5], even x]
[(x,y) | x <- [1..5], odd x, y <- [1..5], even y]

```