Fachrichtung 6.2 — Informatik
Universität des Saarlandes
Tutorenteam der Vorlesung Programmierung 1

Programmierung 1 (Wintersemester 2012/13)

Lösungsblatt 4

(Kapitel 3 und 4)

Aufgabe 4.1 (*Offizielle Übungsaufgabe aus dem Wintersemester 11/12*)
Deklarieren Sie eine Identittsprozedur $''a\ ideq\ :\ ''a\ \rightarrow\ ''a$, deren Typschema auf Typen mit Gleichheit eingeschrnkt ist. Verzichten Sie dabei auf explizite Typangaben.

Lösung 4.1:

```
fun ideq (x) = if x = x then x else x
```

Aufgabe 4.2 (*Offizielle Übungsaufgabe aus dem Wintersemester 11/12*)
Geben Sie zunächst die geschachtelte Paardarstellung von folgenden Listen an. Überlegen Sie sich dann, wie Sie die Liste nur unter Verwendung von *nil* und :: aufschreiben können.

(a) `[1,2,3,4]`

(b) `[1.0,2.0,3.0]`

(c) `[(1,4), (4,1), (6,2), (3,8)]`

(d) `[[4, 2], [3, 2, 1], []]`

Lösung 4.2:

(a) Paardarstellung: $(1, (2, (3, (4, ()))))$
Listendarstellung: $1 :: 2 :: 3 :: 4 :: nil$

(b) Paardarstellung: $(1.0, (2.0, (3.0, ())))$
Listendarstellung: $1.0 :: 2.0 :: 3.0 :: nil$

(c) Paardarstellung: $((1, 4), ((4, 1), ((6, 2), ((3, 8), ()))))$
Listendarstellung: $(1, 4) :: (4, 1) :: (6, 2) :: (3, 8) :: nil$

(d) Paardarstellung: $((4, (2, ())), ((3, (2, (1, ()))), ((), ())))$
Listendarstellung: $(4 :: 2 :: nil) :: (3 :: 2 :: 1 :: nil) :: nil :: nil$

Aufgabe 4.3 (*Offizielle Übungsaufgabe aus dem Wintersemester 11/12*)
Schreiben Sie folgende Prozeduren:

(a) Eine Prozedur $listEvenOdd : int\ list \rightarrow (int * bool)\ list$, welche zu einer Liste von Zahlen folgendes zurckgibt:
$listEvenOdd[x_1, x_2, \ldots, x_n] = [(x_1, b_1), (x_2, b_2), \ldots, (x_n, b_n)]$, wobei b_i jeweils *true* sein soll, wenn x_i gerade ist, ansonsten $false$ (für alle $i \in \{1, \ldots, n\}$).

(b) Eine Prozedur $countEvenOdd : int\ list \rightarrow int * int$, welche zu einer Liste von Zahlen das Tupel (e, o) zurückgibt, sodass e die Anzahl gerader Zahlen in der Liste und o die Anzahl ungerader Zahlen in der Liste angibt.

Lösung 4.3:

(a)
```
fun listEvenOdd nil = nil | listEvenOdd (x::xs) = if x mod 2=0 then ((x, true):: listEvenOdd xs) else ((x, false):: listEvenOdd xs)
```

(b)
```
fun countEvenOdd nil = (0,0) | countEvenOdd (x::xs) = let val (e,n) = countEvenOdd xs in if x mod 2 = 0 then (e+1,n) else (e,n+1) end
```

Aufgabe 4.4 (*Offizielle Übungsaufgabe aus dem Wintersemester 11/12*)
Das Sieb des Eratosthenes ist ein Algorithmus zur Berechnung von Primzahlen bis einschließlich einem gegebenen Index n. Dazu wird eine Liste oder Tabelle von Zahlen von 2 bis n verwendet, aus der nach und nach mehr Zahlen rausgestrichen werden. Solange es noch Elemente in der Liste gibt, betrachtet man jeweils die kleinste Zahl, die in der Liste ist, nimmt sie aus der Liste raus und markiert sie als Primzahl. Dann streicht man alle Vielfachen von ihr aus der Liste. Die Menge der markierten Zahlen ist dann gerade die Menge aller Primzahlen zwischen 2 und n. Im Folgenden soll dieses Prinzip in SML umgesetzt werden.

(a) Schreiben Sie eine Prozedur *listevonzahlen* : $int \to int \to int\ list \to int\ list$, die zu von einem Startindex m an alle Zahlen einschließlich n aufsteigend geordnet in eine Liste schreibt.

(b) Schreiben Sie eine Prozedur *loesche* : $int \to int\ list \to int\ list$, die aus einer gegebenen Liste alle Zahlen entfernt, die durch eine gebene Zahl n teilbar sind.

(c) Schreiben Sie nun eine Prozedur $sieb' : int\ list \to int\ list \to int\ list$, die aus der einer Liste xs das erste Element x entfernt, es an die andere Liste ys anhängt und alle Zahlen aus xs entfernt, die durch x teilbar sind.

(d) Fügen Sie nun die drei Hilfsprozeduren zum Sieb des Erathostenes zusammen.

Lösung 4.4:

(a)
```
fun listevonzahlen m n xs = if m<n then xs else listevonzahlen (m-1) n (m::xs)
```

(b)
```
fun loesche n nil = nil | loesche n (x::xr) = if (x mod n = 0) then loesche n xr else x::(loesche n xr)
```

(c)
```
fun sieb' xs nil = xs | sieb' xs (y::yr) = sieb' (xs@[y]) (loesche y yr)
```

(d)
```
fun siebdeserathostenes n = sieb' nil (listevonzahlen n 2 nil)
```

Aufgabe 4.5 (*Offizielle Übungsaufgabe aus dem Wintersemester 11/12*)
In dieser Aufgabe sollen Sie Ihren bisherigen Punkteschnitt aus den Tests berechnen lassen. Schreiben Sie dazu eine Prozedur *average* : $real\ list \to real$, die mithilfe von $foldl$ das arithmetische Mittel aller Zahlen einer Liste berechnet wie folgt:

(a) Unter Verwendung einer Hilfsprozedur *lengthReal* : $\alpha\ list \to real$, die die Länge einer Liste als *real* ausgibt

(b) Ohne Verwendung von *lengthReal* mithilfe von Faltung.

Lösung 4.5:

(a)
```
fun lengthReal nil = 0.0 |lengthReal (x::xr) = 1.0 + lengthReal xr fun average xr = (foldl op+ 0.0 xr)/(length xr)
```

(b)
```
fun average xr = let val (a, b)= foldl (fn (x, (y, z))=>(y+1.0, z+x)) (0.0, 0.0) xr in b/a end
```

Aufgabe 4.6
Sei folgende Prozedur bereits deklariert:

```
fun f ((2, 4.0)::xr) = 42
  | f [(1, y), (_, 3.0), (_, 4.0)] = 5
  | f [(x, _), (u, w)] = u + x
  | f [(44, 11.0), (12, 3.0)] = 42
  | f (x::xr) = #1(hd xr)
```

Was ergeben die folgenden Prozeduranwendungen?

(a) `f [(1, 5.0), (4711, 3.0), (0815, 4.0)]`

(b) `f [(2, 4.0), (4, 2.0)]`

(c) `f [(2, 4.0)]`

(d) `f [(44, 11.0), (12, 3.0)]`

(e) `f nil`

(f) `f [(2, 4.1)]`

(g) `f [(2, 4.00000000000000001)]`

Lösung 4.6:

(a) $5 : int$

(b) $42 : int$

(c) $42 : int$

(d) $56 : int$

(e) Die Ausnahme $Match$ wird geworfen.

(f) Die Ausnahme *Empty* wird geworfen.

(g) Es wird kein Muster getroffen, deshalb wird eigentlich die Ausnahme *Empty* geworfen. Die in der Vorlesung benutzten SML-Interpreter runden jedoch, weswegen sie zu $42 : int$ auswerten.

Aufgabe 4.7 (*Offizielle Übungsaufgabe aus dem Wintersemester 11/12*)
Sie wurden bei VIVA eingestellt, um einen neuen Algorithmus für den *Love-Generator* zu implementierten. Dieser berechnet die Kompatibilitt zwischen zwei Menschen anhand ihrer Namen und gibt die Beziehungsaussichten in Prozent an. Ihre Aufgabe ist es nun, folgenden, wissenschaftlich höchst fundierten Algorithmus zu implementieren: Schreiben Sie eine Prozedur $loveGen : string * string \to int$, die für jeden der Strings die Ordinalszahlen aufaddiert, daraus die Summe bildet und davon den Rest der Division mit 100 zurückgibt.

(a) Schreiben Sie zunächst eine Hilfsprozedur $characterStrength : string \to int$, die die Summe der Ordinalszahlen aller Buchstaben eines Strings zusammenzählt. Nutzen Sie dafür $foldl$.

(b) Nutzen Sie die Hilfsprozedur *characterStrength*, um *loveGen* wie beschrieben zu implementieren.

(c) Um den Lovegenerator auch international etablieren zu knnen, bentigt VIVA eine angepasste Version, um die Kompatibilitt eines ganzen Harems zu berechnen. Schreiben Sie nun eine Prozedur $loveGenHarem : string\ list \to int$, die die Kompatibilitt eine ganzen Harems berechnet. Nutzen Sie dafr erneut *characterStrength* und $foldl$, jedoch keine sonstigen Hilfsprozeduren.

Lösung 4.7:

(a) `fun characterStrength a = foldl (fn (x, y) => ord x + y) 0 (explode a)`

(b) `fun loveGen (x, y) = (characterStrength x + characterStrength y) mod 100`

(c) `fun loveGenHarem xr = (foldr (fn (x, y)=> characterStrength x + y) 0 xr) mod 100`

Aufgabe 4.8 (*Schriftliche Aufgabe aus dem Wintersemester 11/12*)
Welches Thema der Vorlesung hat Sie bisher am meisten fasziniert oder überrascht? Versuchen Sie den Leser mit Ihrer Faszination für das Thema anzustecken, indem Sie erklären, was Ihre Faszination oder Überraschung ausgelöst hat.

Aufgabe 4.9 (*Schriftliche Aufgabe aus dem Wintersemester 11/12*)
Erklären Sie die folgenden Begriffe ausführlich in eigenen Worten und setzen Sie sie zueinander in Beziehung.

Polymorphe Typisierung: Typschemen, freie und quantifizierte Typvariablen, Instanziierung und Instanzen; polymorphe, monomomorphe und ambige Deklarationen; Spezifikation polymorpher Prozeduren.