McMaster University
Department of Computing and Software
Dr. W. Kahl

SFWR ENG 3E03
Exercise Sheet 4

Design and Selection of Programming Languages

5 October 2006

Exercise 4.1

Assume the following Haskell definitions:

$size = 10$
$square\ n = n * n$

Add a definition for $cube$ with the obvious meaning, and manually perform single-stepped expression evaluation for the expression "$cube\ size - cube\ (size - 2)$".

Exercise 4.2

Haskell has predefined types $Float$ for single-precision floating point numbers (which we ignore in the following) and $Double$ for double-precision floating point numbers.

Standard mathematical functions like

$sqrt, sin, atan :: Double \to Double$

and $pi :: Double$ are also available; $x \wedge k$ stands for x^k if k is natural; $x ** q$ can be used for x^q where both x and q are of type $Double$.

Define the following Haskell functions, with the meanings obvious from their names:

(a) $sphereVolume :: Double \to Double$
(b) $sphereSurface :: Double \to Double$
(c) $centuryToPicosecond :: Integer \to Integer$

Try the last one in C or Java, too; test both, and compare the results

Exercise 4.3

Define the following Haskell functions:

(a) $stutter :: [a] \to [a]$

duplicates each element of its argument lists, e.g.: $stutter\ [1,2,3] = [1,1,2,2,3,3]$

(b) $splits :: [a] \to [([a],[a])]$

delivers for each argument list all possibilities to segment it into non-empty prefix and suffix, e.g.:

$splits\ [1,2,3] = [([1],[2,3]), ([1,2],[3])]$

(The order is irrelevant.)

(c) $rotations :: [a] \to [[a]]$

delivers for each argument list all different results of rotations, each result only once, e.g.:

rotations [1,2,3] = [[1,2,3], [3,1,2], [2,3,1]]

(The order is irrelevant.)

(d) *permutations* :: [a] → [[a]]

delivers for each argument list all different results of permutations, each result only once, e.g.:

permutations [1,2,3] = [[1,2,3], [1,3,2], [2,1,3], [2,3,1], [3,1,2], [3,2,1]]

(The order is irrelevant.)

Exercise 4.4 — Defining Haskell Functions (40% of Midterm 1, 2003)

Define the follwing Haskell functions (the solutions are independent of each other):

(a) *polynomial* :: [*Double*] → *Double* → *Double*

such that for coefficients $c_0, c_1, c_2, \ldots, c_n$ and any x the following holds:

$$\mathit{polynomial}\ [c_0, c_1, c_2, \ldots, c_n]\ x = c_0 + c_1 \cdot x + c_2 \cdot x^2 + \cdots + c_n \cdot x^n$$

e.g.: *polynomial* [3,4,5] 100.0 = 50403.0

Hint: Use Horner's rule:

$$c_0 + c_1 \cdot x + c_2 \cdot x^2 + \cdots + c_n \cdot x^n \;=\; c_0 + x \cdot (c_1 + x \cdot (c_2 + \cdots + x \cdot (c_n) \cdots))$$

(b) *findJump* :: *Integer* → [*Integer*] → (*Integer*, *Integer*)

takes an integer *d* and a list and returns the first pair of **adjacent** elements of the list such that the values of these two elements are farther than *d* apart, e.g.,

findJump 3 [2,3,4,2,5,3,6,2,3,5,4,1,6] = (6,2)

If the list contains no such values, an error is produced.

(c) *suffixes* :: [a] → [[a]]

delivers for each argument list all its suffixes, e.g.:

suffixes [1,2,3,4] = [[1,2,3,4],[2,3,4],[3,4],[4],[]]

(The order is irrelevant.)

(d) *diagonal* :: [[a]] → [a]

interprets its argument as a matrix (represented as in Exercise 2.1), which may be assumed to be square, and returns the main diagonal of that matrix, e.g.:

diagonal [[1,2,3],[4,5,6],[7,8,9]] = [1,5,9]

(e) *isSquare* :: [[a]] → *Bool*

determines whether its argument corresponds to a list-of-lists representation (as in Exercise 2.1) of a ***square*** matrix.

Exercise 4.5 — Haskell Evaluation (30% of Midterm 1, 2003)

Assume the following Haskell definitions to be given:

```
foldr              :: (a -> b -> b) -> b -> [a] -> b
foldr f e []       = e
foldr f e (x:xs) = f x (foldr f e xs)

concat             = foldr (++) []

(||)               :: Bool -> Bool -> Bool  -- Boolean disjunction: or
True  || _         =  True
False || b         =  b

any p = foldr ((||) . p) False

gen f (x,s) = x : gen f (f x s)

foo k n = (k + n, n + 2)
```

Simulate Haskell evaluation for the following expressions (write down the sequence of intermediate expressions):

(a) `foldr (*) 1 [6,7]`

(b) `any (> 0) (gen foo (0,1))`

Exercise 4.6 — Defining Haskell Functions (20% of Midterm 1, 2004)

Define the follwing Haskell functions (the solutions are independent of each other):

(a) $sum :: [\,Integer\,] \rightarrow Integer$

such that $sum\ xs$ evaluates to the sum of all elements of the list xs.

(b) $all :: (\,a \rightarrow Bool\,) \rightarrow [\,a\,] \rightarrow Bool$

such that $all\ \ p\ \ xs$ evaluates to **True** if p considered as a predicate holds for all elements of x, and to **False** if there is at least one element in xs for which p does not hold.

E.g., $all\ (>1)\ [2..10] = \mathbf{True}$

(c) $selMod :: Integer \rightarrow [\,Integer\,] \rightarrow [\,Integer\,]$

such that $selMod\ k\ xs$ selects from the list xs all those elements that are equivalent to k modulo $k + 1$, e.g.,

$selMod\ 2\ [2,3,8,1,2,5] = [2,8,2,5]$

(d) $sources :: Eq\ a \Rightarrow [\,(a,a)\,] \rightarrow [\,a\,]$

such that $sources\ ps$ returns the *sources* of the graph ps.

Here, the list ps of pairs is considered as representing a simple graph by representing each edge from node x to node y by the pair (x, y).

The *context* "$Eq\ a \Rightarrow$" just means that you may use the equality test for elements of type a, i.e., $(==) :: a \rightarrow a \rightarrow Bool$.

Example: $sources\ [(2,3), (3,4), (1,4), (1,5), (2,5)] = [2,1]$

(The order is irrelevant.)