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

SFWR ENG 3E03
Exercise Sheet 3

Design and Selection of Programming Languages

28 September 2005

Exercise 1.1

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 1.2

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 -> Double
```

and `pi :: Double` are also available; `x ^ 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 -> Double`

(b) `sphereSurface :: Double -> Double`

(c) `centuryToPicosecond :: Integer -> Integer`
Try the last on in C or Java, too; test both, and compare the results

Exercise 2.2

Define the following Haskell functions:

(a) *stutter* :: [*a*] → [*a*]
duplicates each element of its argument lists, e.g.:
stutter [1,2,3] = [1,1,2,2,3,3]

(b) *splits* :: [*a*] → [([*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*] → [[*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 3.1 — 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:

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

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 2.3 — 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))`