Software Workshop Haskell: Exercises 1

Antoni Diller

2 September 2010

1 Functions Involving Integers

(1.1) Write a function *sotls* (sum of two largest squares) so that *sotls* x y z is the sum of the squares of the two largest integers x, y and z.

(1.2) Define a function *sumsq* which takes an integer n as its argument and returns the sum of the squares of the first n integers. That is to say,

$$\mathit{sumsq}\ n = 1^2 + 2^2 + 3^2 + \ldots + n^2.$$

(1.3) Define the factorial function *fact* which behaves as follows: *fact* $n = 1 \times 2 \times \ldots \times n$.

(1.4) Define a function *comb* which takes positive integers n and m and returns the number of combinations of n objects taken m at a time. That is to say,

$$\mathit{comb}\ n\ m = \frac{n!}{m! \times (n-m)!}.$$

Note that *comb* is not defined if $n < m$. When this happens your definition should return an error message.

(1.5) Define a function *mygcd* which takes positive integers x and y as arguments and returns the greatest common divisor of x and y as its value. Note that *mygcd* should return an error message when both of its arguments are zero.

(1.6) Write a program to determine whether or not a given number is prime, that is to say, has no divisors other than 1 and itself. Call your function *prime*.

(1.7) A perfect number is one which is equal to the sum of its divisors, excluding itself, but including 1. Thus, 6 is perfect because $6 = 1 + 2 + 3$ and each of 1, 2 and 3 divide 6. Write a function *perfect* which tests its single argument for perfection.

(1.8) Suppose that you have a function *coin* which is such that *coin* i is the value of the ith coin in some currency. For example, in the United Kingdom we have:

$$\begin{aligned}
coin\ 1 &= 1,\\
coin\ 2 &= 2,\\
coin\ 3 &= 5,\\
coin\ 4 &= 10,\\
coin\ 5 &= 20,\\
coin\ 6 &= 50,\\
coin\ 7 &= 100,\\
coin\ 8 &= 200.
\end{aligned}$$

Write a function *countways* which is such that *countways* n m returns the number of different ways to make change from an amount m using n coins (of any value).

(1.9) An abundant number is a natural number whose distinct proper factors have a sum exceeding that number. Thus, 12 is abundant because $1+2+3+4+6 > 12$. Write a Boolean-valued function *abundant* which tests whether or not a number is abundant.

(1.10) Two numbers are amicable if each is the sum of the distinct proper factors of the other. For example, 220 and 284 are amicable because the factors of 284 are 1, 2, 4, 71 and 142 and these add up to 220 and because the factors of 220 are 1, 2, 4, 5, 10, 11, 20, 22, 44, 55 and 110 and these add up to 284. Write a function *amicable* which tests whether or not two distinct numbers are amicable.

(1.11) The least common multiple of a set of numbers is the smallest number that is exactly divisible by all of the numbers in the set. For example, the least common multiple of 3, 5 and 10 is 30. Write a function *lcm3* which is such that *lcm3* i j k is the least common multiple of the three numbers i, j and k,

2 Lists

(2.1) Define a function *productList* :: [*Int*] $\rightarrow$ *Int* which returns the product of a list of integers. You should take the product of the empty list to be 1.

(2.2) Define a function *myand* :: [*Bool*] $\rightarrow$ *Bool* which returns the conjunction of a list. Informally,

$$myand\ [e_1, e_2, \ldots, e_i] = e_1\ \&\&\ e_2\ \&\&\ \ldots\ \&\&\ e_i.$$

The conjunction of an empty list should be *True*.

(2.3) Define a function *concatList* :: [[*Int*]] $\to$ [*Int*] which flattens a list of lists of integers into a single list of integers. For example,

$$concatList\ [[3,4],[],[31,3]] = [3,4,31,3].$$

Informally,

$$concatList\ [e_1, e_2, \ldots, e_i] = e_1 \ +\!\!+\ e_2 \ +\!\!+\ \ldots \ +\!\!+\ e_i.$$

(2.4) Define the function *while* which is such that *while pred xs* returns the longest initial segment of the list *xs* all of whose elements satisfy the Boolean-valued function *pred*. For example,

$$while\ even\ [2,4,8,3,4,8,6] = [2,4,8].$$

(2.5) The function *iSort* (insertion sort) is defined as follows:

```
iSort :: [Int] -> [Int]
iSort []     = []
iSort (x:xs) = ins x (iSort xs)

ins  :: Int -> [Int] -> [Int]
ins x [] = [x]
ins x (y:ys)
  | x <= y    = x:y:ys
  | otherwise = y:ins x ys
```

Use the function *iSort* to define two functions, *minList* and *maxList*, which find the minimum and maximum elements of a non-empty list of integers.

(2.6) Define the functions *minList* and *maxList*, which return the minimum and maximum elements of a non-empty list of integers, respectively, without using *iSort* or any other sorting function.

(2.7) Using the function *iSort* defined in question (2.5) redefine the function *ins* so that the list is sorted in descending order.

(2.8) Using the function *iSort* defined in question (2.5) redefine the function *ins* so that, in addition to outputting a list in ascending order, duplicates are removed. For example, *iSort* $[2,1,4,1,2] = [1,2,4]$.

(2.9) Define the function *memberNum* :: [*Int*] $\to$ *Int* $\to$ *Int* such that *memberNum xs x* returns the number of times that x occurs in the list *xs*. For example,

$$memberNum\ [2,1,4,1,2]\ 2 = 2.$$

(2.10) The function *member* :: [*Int*] $\rightarrow$ *Int* $\rightarrow$ *Bool* has the property that *member xs x* returns *True* if x occurs in the list *xs* and it returns *False* if x does not occur in the list *xs*. Give a definition of *member* which uses the function *memberNum* that you defined as the answer to question (2.9).

(2.11) Redefine the function *member* of question (2.10) so that it no longer makes use of *memberNum* (from question (2.9)).

(2.12) Using pattern matching with : (cons), define a function *rev2* that reverses all lists of length 2, but leaves all other lists unchanged.

(2.13) Define a function *position* which takes a number i and a list of numbers *xs* and returns the position of i in the list *xs*, counting the first position as 1. If i does not occur in *xs*, then *position* returns 0.

(2.14) Define a function *element* which takes a list *xs* and a positive integer i and returns the ith member of *xs*. Assume that the list *xs* is at least of length i.

(2.15) Define a function *segments* which takes a finite list *xs* as its argument and returns the list of all the segments of *xs*. (A segment of *xs* is a selection of adjacent elements of *xs*.) For example, *segments* $[1, 2, 3] = [[1, 2, 3], [1, 2], [2, 3], [1], [2], [3]]$.

(2.16) A partition of a positive integer n is a representation of n as the sum of any number of positive integral parts. For example, there are 7 partitions of the number 5: $1+1+1+1+1$, $1+1+1+2$, $1+1+3$, $1+2+2$, $1+4$, $2+3$ and 5. Define a function *parts* which returns the list of distinct partitions of an integer n. For example, *parts* $4 = [[1, 1, 1, 1], [1, 1, 2], [1, 3], [2, 2], [4]]$.

(2.17) A segment *ys* of a list *xs* is said to be flat if all the elements of *ys* are equal. Define *llfs* such that *llfs xs* is the length of the longest flat segment of *xs*.

(2.18) A list of numbers is said to be steep if each element of the list is at least as large as the sum of the preceding elements. Define a function *llsg* such that *llsg xs* is the length of the longest steep segment of *xs*.

(2.19) Define a function *llsq* such that *llsq xs* is the length of the longest steep subsequence of *xs*.

(2.20) Given a sequence of positive and negative integers define a function *msg* which returns the minimum of the sums of all the possible segments of its argument.