Haskell Answers 4:ZF-expressions

Antoni Diller

4 August 2011

(1) Define a function *unique* :: [*Int*] → [*Int*] which returns the list of numbers in a list that occur exactly once in that list. For example, *unique* $[2, 4, 2, 1, 4] = [1]$.

```
unique :: [Int] -> [Int]
unique xs = [ x | x <- xs, memberNum xs x == 1 ]
```

(2) Redefine the function *member*, explained in question (2.10), using a list comprehension (that is to say, a ZF-expression).

```
member2 :: [Int] -> Int -> Bool
member2 xs y = or [y == x | x <- xs]
```

(3) Consider the list of numbers *wirth* defined as follows:

(i) 1 is a member of *wirth*.

(ii) If x is a member of *wirth*, then so are $2 \times x + 1$ and $3 \times x + 1$.

(iii) These are the only elements of *wirth*.

Give a definition of *wirth* which produces the elements in increasing order.

```
wirth :: [Integer]
wirth = 1 : merge [ 2*x + 1 | x <- wirth ] [ 3*x + 1 | x <- wirth ]

merge :: Ord a => [a] -> [a] -> [a]
merge [] ys = ys
merge xs [] = xs
merge (x:xs) (y:ys)
  | x == y = x : merge xs ys
  | x < y  = x : merge xs (y:ys)
  | x > y  = y : merge (x:xs) ys
```

(4) Define the function *sumsq*, which takes an integer n as its argument and returns the sum of the squares of the first n integers, using a ZF-expression.

```
sumsq3 :: Integral a => a -> a
sumsq3 n = sum [ i*i | i <- [1..n] ]
```

(5) Using a ZF-expression define a Boolean-valued function *prime* such that $prime\ i = True$ if and only if i is a prime number.

```
prime :: Integral a => a -> Bool
prime i = and [ i `rem` j /= 0 | j <- [2..(i-1)] ]
```

(6) Using a ZF-expression define the function *factors* such that *factors* n is the list of all the factors of n in increasing order. For example, *factors* $6 = [1, 2, 3, 6]$.

```
factors :: Integral a => a -> [a]
factors n = [ i | i <- [1..n], n `rem` i == 0 ]
```

(7) A number n is said to be perfect if the factors of n, including 1 but excluding n, add up to n. For example, 6 is perfect becasue $6 = 1 + 2 + 3$. Using the function *factors* from question (6) define the list of all perfect numbers.

```
perfect :: Integral a => a -> Bool
perfect n = sum (factors n) == 2 * n

perfectList :: [Integer]
perfectList = [ i | i <- [1..], perfect i ]
```

(8) Define a function *perms* which takes a finite list *xs* as its argument and returns the list of all the permutations of *xs*.

```
(\\) :: Eq a => [a] -> [a] -> [a]
(\\) = foldl del where del [] _      = []
                       del (x:xs) y
                         | x == y    = xs
                         | otherwise = x:del xs y

perms :: Eq a => [a] -> [[a]]
perms [] = [[]]
perms xs = [ y:ys | y <- xs, ys <- perms (xs \\ [y]) ]
```

(9) Define a function *isSquare* which tests to see if a positive number is equal to the square of some integer. For example,

$$\begin{aligned} \mathit{isSquare}\ 7 &= \mathit{False}, \\ \mathit{isSquare}\ 16 &= \mathit{True}. \end{aligned}$$

```
squares :: [Integer]
squares = [ i*i | i <- [1..] ]

auxisSquare :: Ord a => a -> [a] -> Bool
auxisSquare x (s:ss)
  | x == s = True
  | x > s  = auxisSquare x ss
  | x < s  = False

isSquare :: Integer -> Bool
isSquare x = auxisSquare x squares
```

(10) Using a single list-comprehension define the *gcd* function. Call your function *gcdlist*. The *gcd* function can be defined like this:

$$\begin{aligned} gcd(i, j) &= i, \text{if } j = 0 \\ &= gcd(j, i \mod j), \text{if } j \neq 0 \end{aligned}$$

(11) Define the function *power* using a list-comprehension, where

$$\mathit{power}\ x\ n = x^n.$$

Here, x can be any number, but n must be an integer (either positive, negative or zero).

(12) The function *iteraten* is defined as follows:

```
iteraten :: Int -> (a -> a) -> a -> a
iteraten 0 f x = x
iteraten n f x = iteraten (n - 1) f (f x)
```

Define a function *limit* such that $limit\ f\ x = iteraten\ n\ f\ x$ where n is the least number such that $iteraten\ n\ f\ x = iteraten\ (n+1)\ f\ x$

```
itlist f x = [ iteraten n f x | n <- [0..] ]
test (x:y:xs)
  | x == y    = x
  | otherwise = test (y:xs)
limit f x = test (itlist f x)
```