(1) Define a function \( \text{unique} :: \text{[Int]} \to \text{[Int]} \) which returns the list of numbers in a list that occur exactly once in that list. For example, \( \text{unique} \{2, 4, 2, 1, 4\} = \{1\} \).

\[
\text{unique} :: \text{[Int]} \to \text{[Int]} \\
\text{unique} \; \text{x} = \{ \; \text{x} \mid \text{x} \leftarrow \text{x}, \; \text{memberNum} \; \text{x} \; \text{x} = 1 \; \}
\]

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

\[
\text{member2} :: \text{[Int]} \to \text{Int} \to \text{Bool} \\
\text{member2} \; \text{x} \; \text{y} = \text{or} \; \{ \text{y} = \text{x} \mid \text{x} \leftarrow \text{x} \}
\]

(3) Consider the list of numbers \( \text{wirth} \) defined as follows:

(i) 1 is a member of \( \text{wirth} \).
(ii) If \( x \) is a member of \( \text{wirth} \), then so are \( 2 \times x + 1 \) and \( 3 \times x + 1 \).
(iii) These are the only elements of \( \text{wirth} \).

Give a definition of \( \text{wirth} \) which produces the elements in increasing order.

\[
\text{wirth} :: \text{[Integer]} \\
\text{wirth} = 1 : \text{merge} \; \{ 2 \times \text{x} + 1 \mid \text{x} \leftarrow \text{wirth} \} \; \{ 3 \times \text{x} + 1 \mid \text{x} \leftarrow \text{wirth} \}
\]

\[
\text{merge} :: \text{Ord a} => \text{[a]} \to \text{[a]} \to \text{[a]} \\
\text{merge} \; [] \; \text{ys} = \text{ys} \\
\text{merge} \; \text{xs} \; [] = \text{xs} \\
\text{merge} \; (\text{x}:\text{x}s) \; (\text{y}:\text{ys}) \\
\quad \mid \text{x} == \text{y} = \text{merge} \; \text{x}s \; \text{ys} \\
\quad \mid \text{x} < \text{y} = \text{merge} \; \text{x}s \; (\text{y}:\text{ys}) \\
\quad \mid \text{x} > \text{y} = \text{merge} \; (\text{x}:\text{x}s) \; \text{ys}
\]

1
(4) Define the function \texttt{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.

\begin{verbatim}
sumsq3 :: Integral a => a -> a
sumsq3 n = sum [ i*i | i <- [1..n] ]
\end{verbatim}

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

\begin{verbatim}
prime :: Integral a => a -> Bool
prime i = and [ i \text{ 'rem' } j /= 0 | j <- [2..(i-1)] ]
\end{verbatim}

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

\begin{verbatim}
factors :: Integral a => a -> [a]
factors n = [ i | i <- [1..n], n \text{ 'rem' } i == 0 ]
\end{verbatim}

(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 \texttt{factors} from question (6) define the list of all perfect numbers.

\begin{verbatim}
perfect :: Integral a => a -> Bool
perfect n = sum (factors n) == 2 * n

perfectList :: [Integer]
perfectList = [ i | i <- [1..], perfect i ]
\end{verbatim}

(8) Define a function \texttt{perms} which takes a finite list \( xs \) as its argument and returns the list of all the permutations of \( xs \).
\( (\_\_\_\_ \_\text{Eq } a \Rightarrow [a] \rightarrow [a] \rightarrow [a] \)
\( (\_\_\_\_ = \text{foldl del where del } [] \_ = [] \)
\( \quad \text{del } (x:xs) \ y \)
\( \quad \mid x == y = xs \)
\( \quad \mid \text{otherwise } = \text{del } xs \ y \)
\( \quad \quad \)

\( \text{perms } :: \text{Eq } a \Rightarrow [a] \rightarrow [[a]] \)
\( \text{perms } [] = [[]] \)
\( \text{perms } xs = [ y:ys | y <- xs, ys <- \text{perms } (xs \setminus [y]) ] \)

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

\[
\begin{align*}
\text{isSquare } 7 &= \text{False}, \\
\text{isSquare } 16 &= \text{True}.
\end{align*}
\]

\( \text{squares } :: [\text{Integer}] \)
\( \text{squares } = [ i*i | i <- [1..] ] \)

\( \text{auxisSquare } :: \text{Ord } a \Rightarrow a \rightarrow [a] \rightarrow \text{Bool} \)
\( \text{auxisSquare } x (s:ss) \)
\( \quad | x == s = \text{True} \)
\( \quad | x > s = \text{auxisSquare } x ss \)
\( \quad | x < s = \text{False} \)

\( \text{isSquare } :: \text{Integer } \rightarrow \text{Bool} \)
\( \text{isSquare } x = \text{auxisSquare } x \text{ squares} \)

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

\[
\begin{align*}
\text{gcd}(i, j) &= i, \text{if } j = 0 \\
&= \text{gcd}(j, i \ mod \ j), \text{if } j \neq 0
\end{align*}
\]

(11) Define the function \( \text{power} \) using a list-comprehension, where

\[
\text{power } x \ n = x^n.
\]

Here, \( x \) can be any number, but \( n \) must be an integer (either positive, negative or zero).
The function \textit{iteraten} is defined as follows:

\begin{verbatim}
iteraten :: Int -> (a -> a) -> a -> a
iteraten 0 f x = x
iteraten n f x = iteraten (n - 1) f (f x)
\end{verbatim}

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

\begin{verbatim}
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)
\end{verbatim}