```haskell
import Data.Char
-- Problem 1
volSphere :: Float -> Float
volSphere r = (4/3) * pi * (r^3)
-- Problem 2
pow :: Float -> Int -> Float
pow n 1 = n
pow n p = n * (pow n (p - 1))
-- Problem 3
maxFinder :: [Int] -> Int
maxFinder l = findMaxHelper (-99) l

findMaxHelper :: Int -> [Int] -> Int
findMaxHelper m [] = m
findMaxHelper m l
        | m < (head l) = findMaxHelper (head l) (tail l)
        | otherwise = findMaxHelper m (tail l)
-- Problem 4
wsum :: [Int] -> [Int] -> [Int]
wsum [] [] = []
wsum a b = zipWith (+) [(2 * head a)] [(head b)] ++ wsum (tail a) (tail b)
-- Problem 5
repeatNum :: (Eq a) => a -> [a] -> Int
repeatNum e l = repeatNumHelper 0 e l

repeatNumHelper :: (Eq a) => Int -> a -> [a] -> Int
repeatNumHelper n e [] = n
repeatNumHelper n e l
        | e == (head l) = repeatNumHelper (n + 1) e (tail l)
        | otherwise = repeatNumHelper n e (tail l)
-- Problem 6
split :: [a] -> Int -> [[a]]
split l n = j [headAt n l, tailAt n l]

headAt :: Int -> [a] -> [a]
headAt 0 l = []
headAt n l = (head l) : headAt (n - 1) (tail l)

tailAt :: Int -> [a] -> [a]
tailAt 0 l = l
tailAt n l = tailAt (n - 1) (tail l)
-- Problem 7
natSum :: Int
natSum = natSumHelper 1000

natSumHelper :: Int -> Int
natSumHelper 0 = 0
natSumHelper n
        | (n `mod` 3) == 0 = n + natSumHelper (n - 1)
        | (n `mod` 5) == 0 = n + natSumHelper (n - 1)
        | otherwise = natSumHelper (n - 1)
-- Problem 8 A
decoder ::  (a -> t) -> (a -> t) -> [a] -> [t]
decoder f1 f2 [] = []
decoder f1 f2 s = decoderHelper f1 f2 s (length s)

decoderHelper :: (Integral a) => (a1 -> t) -> (a1 -> t) -> [a1] -> a -> [t]
decoderHelper f1 f2 [] 0 = []
decoderHelper f1 f2 s i
        | even i = (f1 (head s)) : decoderHelper f1 f2 (tail s) (i - 1)
        | otherwise = (f2 (head s)) : decoderHelper f1 f2 (tail s) (i - 1)

rot13 x = chr(((ord x) - 96 + 13) `mod` 26 + 96)
```

```haskell
rot7  x = chr(((ord x) - 96 + 7 ) `mod` 26 + 96)
rot19 x = chr(((ord x) - 96 + 19) `mod` 26 + 96)
-- Problem 9
{- For problem 8, I would have done a method with the signature decoder(Interface a, Interface b, String s).
-- and when calling, it would receive an instance of the implementing child class with a different
-- algorithm to apply for evens and odd characters.

-- Ex  String decoder(new Rot19(), new Rot13(), "xxxxxxxx") {
                        for (...) {
                                if (even) { ns += a.operation(s.charAt(i)); }
                                if (odd) { ns += b.operation(s.charAt(i)); }
                        }
                        return ns;
           }
--
-- But of course Haskell makes it easier since I don't have to worry about having to create a child
-- class for every single operation that I would like to perform on each character.-}
```

```
DxExWxExY@Laptopexe: /mnt/c/Users/DxExWxExY/Google Drive/Documents Backup/CS Prog Lang/Haskell
Ok, modules loaded: Main.
*Main> volSphere 5
523.5988
*Main> pow 2 3
8.0
*Main> max
max        maxBound   maxFinder  maximum
*Main> maxFinder [1,9,8,7,6,8,9,99,2]
99
*Main> maxFinder [1,9,8,7,6,8,9,99,200]
200
*Main> maxFinder [1,9,8,7,699,8,9,99,200]
699
*Main> wsum [1,1,1] [1,1,1]
[3,3,3]
*Main>
*Main> repeatNum 1 [1,1,1,1,1,1,1,155,4,300,9,1]
8
*Main> split [1,2,3,4,5,6,7,8,9,00] 5
[[1,2,3,4,5],[6,7,8,9,0]]
*Main> natSum
234168
*Main> decoder rot19 rot13 "jlvyqhyfnuxvipp"
"weirdalyankovic"
*Main>
```