```haskell
greeting = "Hello, world!"

-- problem 1
largest::String->String->String
largest x y =
   if (length x) >= (length y)
      then x
      else y

-- problem 2
{-
Since Haskell is left associative the recursive calls to reflect will never get the
chance to increment or decrement the num input.  To fix this we change the num value
within paranthesis as seen below
-}
reflect::Integer->Integer
reflect 0 = 0
reflect num
   | num < 0 = (-1) + reflect (num+1)
   | num > 0 = 1 + reflect (num-1)

-- problem 3a
all_factors::Integer->[Integer]
all_factors x = [y | y <- [1..x], (mod x y) == 0]

-- problem 3b
perfect_numbers = [y | y <- [2..], (sum (init(all_factors y))) == y]

-- problem 4.1
is_even::Integer->Bool
{-
is_even x =
   if x == 0
      then True
      else is_odd (x-1)
-}

is_odd::Integer->Bool
{-
is_odd x =
   if x == 0
      then False
      else is_even (x-1)
-}

--problem 4.2
{-
is_even x
   | x == 0 = True
   | otherwise = is_odd (x-1)

is_odd x
   | x == 0 = False
   | otherwise = is_even (x-1)
-}
```

```haskell
--problem 4.3
is_even 0 = True
is_even x = is_odd (x-1)

is_odd 0 = False
is_odd x = is_even (x-1)

--problem 5
count_occurrences [] _ = 1
count_occurrences x [] = 0
count_occurrences x y =
    if head x == head y
        then (count_occurrences (tail x) (tail y)) + (count_occurrences x (tail y))
        else count_occurrences x (tail y)
```