Functional Programming Assignment 1 - Extension

Find the nth Fibonacci number, and generate a list of Fibonacci numbers up to n, using a formula that does not involve calculating each number before it:

$$F(n) = \frac{(\varphi)^n - (-\frac{1}{\varphi})^n}{\sqrt{5}}$$

This makes printing long lists of Fibonacci numbers much faster to do. For comparison, I have also included the "standard" method of computing the Fibonacci sequence.

Intial Declaration

```
module Extension where

import Control.Monad
import System.Environment
import Numeric
```

Declaring the Golden Ratio as a global variable so I can use it multiple times if needed

```
goldRat :: Double
goldRat = ((1 + sqrt 5) / 2)
```

Main function "run". Once the file has compiled, enter "run" into the console and follow the prompts. If you do not enter a valid number, the program throws an error. I couldn't figure out how to make the program re-prompt the user if the input is invalid, but that was the goal.

```
run = do
putStrLn "To find the nth number of the Fibonacci Sequence, please enter n now:"
n <- getLine
let num = read n :: Integer
putStrLn ("Fibonacci Number" ++ n ++ " = " ++ show (fibNum num))
putStrLn ("List of Fibonacci Numbers up to and including number" ++ n ++ ":" ++ show (fibList num))
```

"fibNum" uses the formula at the top of the page to find the nth Fibonacci number.

```
fibNum :: Integer -> Integer
fibNum x = round (((goldRat ^ x) - ((-(1/goldRat))^x)) / sqrt 5)
```

"fibList" uses "fibNum" to quickly generate a list of n Fibonacci numbers.

```
fibList :: Integer -> [Integer]
fibList a = map fibNum [0 .. a]
```

The following functions are the "old" method of checking Fibonacci numbers. To find the nth Fibonacci number, enter "oldFib n", where n is any number you wish to check (numbers above 30 take a long time to calculate). To generate a list of n Fibonacci numbers, enter "oldFibList n" where n is any number you wish to check.

```
oldFib :: Integer -> Integer
oldFib 0 = 0
oldFib 1 = 1
oldFib b = oldFib(b-1) + oldFib(b-2)
```

```
oldFibList :: Integer -> [Integer]
oldFibList c = map oldFib [0 .. c]
```