Suryanarayan.B

CB.EN.U4CSE19056

Assignment-2

1) gcdiv :: Int -> Int -> Int

gcdiv a 0 = a

gcdiv a b = gcd b (a `mod` b)

main = do

    print(" enter dividend and divisor : ")

    x <- getLine

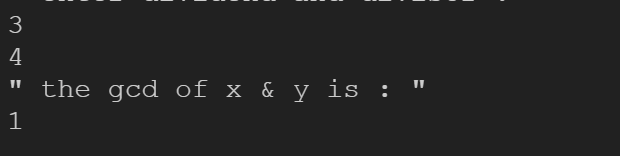
    y <- getLine

    let a = read x :: Int

    let b = read y :: Int

    print(" the gcd of x & y is : ")

    print(gcdiv a b)



2)

fibonacci :: (Integral a) => a -> a

fibonacci 0 = 0

fibonacci 1 = 1

fibonacci n = fibonacci (n - 1) + fibonacci (n - 2)

main = do

    putStrLn("ENTER THE TWO NUMBER : ")

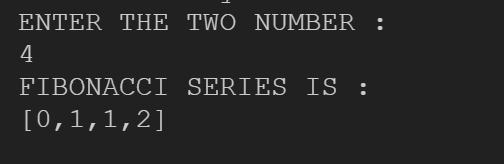
    a <- getLine

    let k = read a :: Int

    putStrLn("FIBONACCI SERIES IS : ")

    let n = take k [fibonacci n | n <- [0..]]

    print(n



3) pro :: ([Int]) -> Int

pro (a) = do

    if a /= [] then

        head a\*pro(drop 1 a)

    else 1

main = do

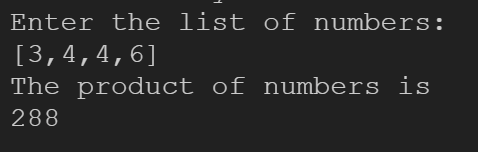
    putStrLn("Enter the list of numbers:")

    num <- getLine

    let a = read num :: [Int]

    putStrLn("The product of numbers is ")

    print(pro a)



4) rev :: Int -> Int

rev 0 = 0

rev n = firstDigit + 10 \* (rev $ n - firstDigit \* 10^place)

 where

 n' = fromIntegral n

 place = (floor . logBase 10) n'

 firstDigit = n `div` 10^place

main = do

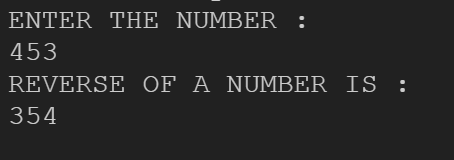
    putStrLn("ENTER THE NUMBER : ")

    a <- getLine

    let k = read a :: Int

    putStrLn("REVERSE OF A NUMBER IS : ")

    print(rev k)



5)

natSum :: Integer -> Integer

natSum 0 = 0

natSum n = n + natSum (n - 1)

main = do

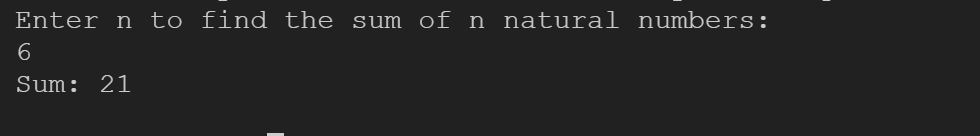
 putStrLn ("Enter n to find the sum of n natural numbers: ")

 inpStr <- getLine

 let a = (read inpStr) :: Integer

 let result = natSum(a)

 putStrLn ("Sum: " ++ show (result))



6)

sum\_d :: Int -> Int

sum\_d 0 = 0

sum\_d n = n`mod`10 + sum\_d(n `div` 10)

main = do

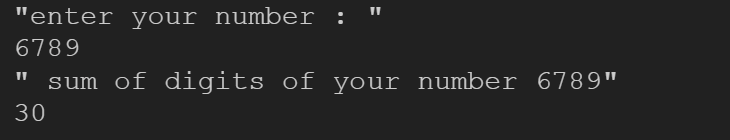
    print("enter your number : ")

    x <- getLine

    let n = read x :: Int

    print(" sum of digits of your number " ++ x)

    print(sum\_d n)



7)

digits::Integer->[Int]

digits n = map(\x ->read[x]::Int)(show n )

countRecursion::Eq a => [a] ->a ->Int

countRecursion [] find = 0

countRecursion (x:xs) find

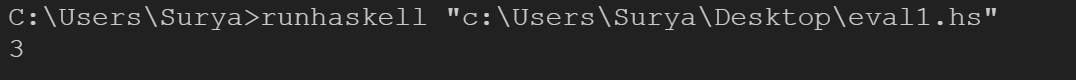
    | find ==x =1+(countRecursion xs find)

    | otherwise = countRecursion xs find

main = do

    let a =digits(1000333)

    print(countRecursion a 0)



8)

pow :: Int -> Int -> Int

pow b 0 = 1

pow b n = b \* (pow b (n-1))

main = do

    print("enter your base and power")

    x <-getLine

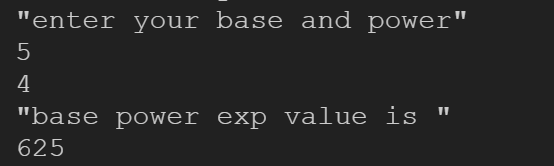
    y <-getLine

    let b = read x :: Int

    let n = read y :: Int

    print("base power exp value is ")

    print(pow b n)



9)

pro :: Int -> Int -> Int

pro a 0 = 0

pro a b = a + (pro a (b-1))

main = do

    print("enter your two numbers")

    x <-getLine

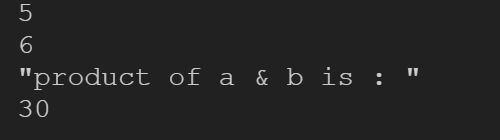
    y <-getLine

    let a = read x :: Int

    let b = read y :: Int

    print("product of a & b is : ")

    print(pro a b)



10)

hail :: Int -> IO()

hail n=do

 print(n)

 if n==1 then print "end of the list"

 else if n `mod` 2==0 then hail(div n 2)

 else hail(n\*3 + 1)

main=do

    putStrLn "Enter n value"

    x <- getLine

    let n = read x :: Int

    print("hail stone series is :")

    hail n

