Haskell Assignment

Q1:

Code:

main = do

    let aline1 = "706 B, Avinashi Road"

    let aline2 = "Pappanaickenpalayam"

    let city = "Coimbatore"

    let state = "Tamil Nadu"

    putStrLn $ "My address is : " ++ aline1 ++ ", " ++ aline2 ++ ", " ++ city ++ ", " ++ state

output:



Q2

Code:  
main :: IO ()

main = do

    let num1 = 10

    let num2 = 20

    let sum = num1 + num2

    putStrLn $ "The sum is: " ++ show sum

    let product = num1\*num2

    let quotient = num2 `div` num1

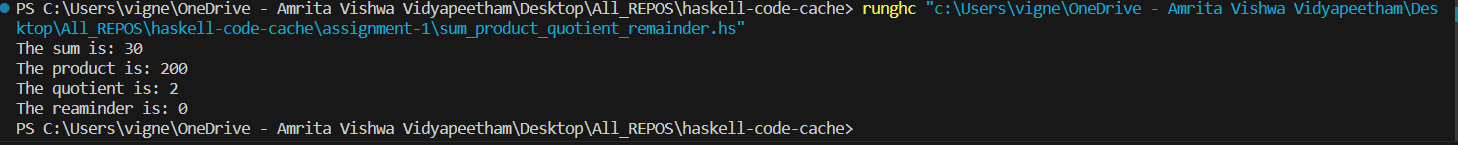
    let remainder = num2 `mod` num1

    putStrLn $ "The product is: " ++ show product

    putStrLn $ "The quotient is: " ++ show quotient

    putStrLn $ "The reaminder is: " ++ show remainder

output:



Q3

Code:

main :: IO ()

main = do

    putStrLn $  "Enter a value in kilometers:"

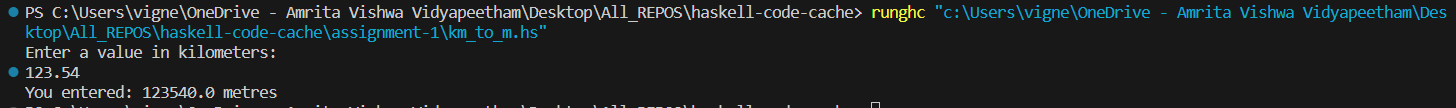
    kmStr <- getLine

    let km = read kmStr :: Double

        metres = km \* 1000

    putStrLn ("You entered: " ++ show metres ++ " metres")

output:



Q4

Code:

main :: IO()

main = do

    putStrLn $ "Enter temperature in degree celsius: "

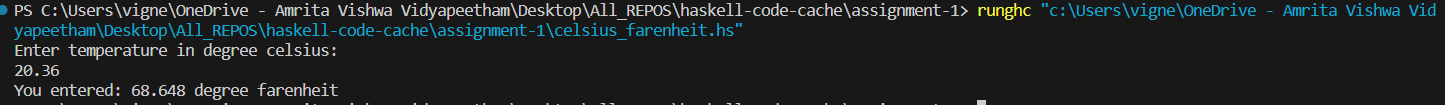
    celsiusStr <- getLine

    let celsius = read celsiusStr :: Double

    let farenheit = (celsius \* 9/5) + 32

    putStrLn ("You entered: " ++ show farenheit ++ " degree farenheit")

Output:



Q5

Code:

main :: IO()

main = do

    putStrLn $ "Enter a value in feet: "

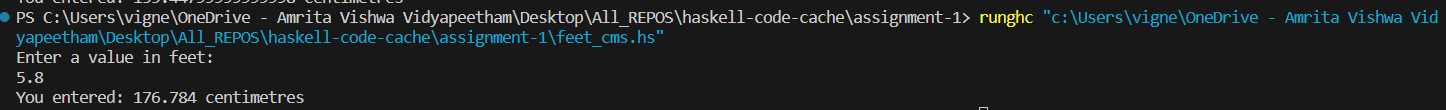
    feetStr <- getLine

    let feet = read feetStr :: Double

    let centimetres = feet \* 30.48

    putStrLn $ "You entered: " ++ show centimetres ++ " centimetres"

Output:



Q6

Code:

main :: IO()

main = do

    putStrLn $ "Enter a number: "

    numberStr <- getLine

    let number = read numberStr :: Double

    let square = number \* number

    let cube = number \* number \* number

    putStrLn $ "You entered: " ++ show number ++ " square: " ++ show square ++ " cube: " ++ show cube

    let sqrt = number \*\* 0.5

    putStrLn $ "You entered: " ++ show number ++ " square root: " ++ show sqrt

output:



Q7

Code:

main :: IO()

main = do

    putStrLn $ "Enter principal amount: "

    princiStr <- getLine

    putStrLn $ "Enter rate of interest: "

    rateStr <- getLine

    putStrLn $ "Enter time period: "

    timeStr <-getLine

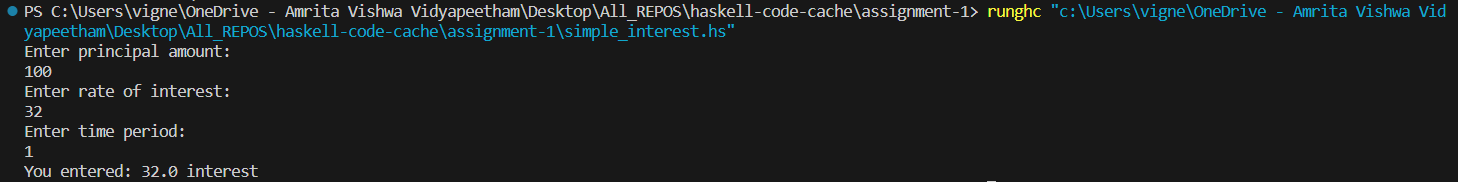
    let princi = read princiStr :: Double

    let rate = read rateStr :: Double

    let time = read timeStr :: Double

    let interest = (princi \*  rate \* time) / 100

    putStrLn $ "You entered: " ++ show interest ++ " interest"

Output:  


Q8

Code: calculateCompoundInterest :: Double -> Double -> Double -> Double

calculateCompoundInterest principal rate time = principal \* (1 + rate/100) \*\* time

main :: IO ()

main = do

    putStrLn "Enter principal amount:"

    principalInput <- getLine

    let principal = readDouble principalInput

    putStrLn "Enter interest rate:"

    rateInput <- getLine

    let rate = readDouble rateInput

    putStrLn "Enter time (in years):"

    timeInput <- getLine

    let time = readDouble timeInput

    case (principal, rate, time) of

        (Just p, Just r, Just t) -> do

            let compoundInterest = calculateCompoundInterest p r t

            putStrLn $ "Compound Interest: " ++ show compoundInterest

        \_ -> putStrLn "Invalid input. Please enter valid numbers."

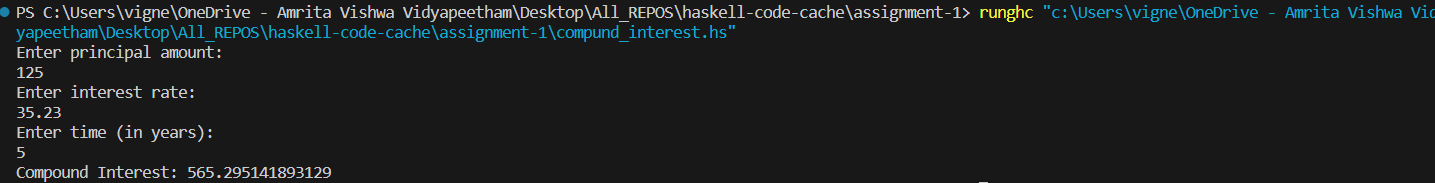
readDouble :: String -> Maybe Double

readDouble str = case reads str of

    [(x, "")] -> Just x

    \_ -> Nothing

Output:



Q9

Code:

main :: IO()

main = do

    putStrLn $ "Enter value for radius: "

    radiusStr <- getLine

    let radius = read radiusStr :: Double

    let area = 4 \* pi \* (radius\*\*2.0)

    let volume = (4/3) \* pi \* (radius\*\*3.0)

    putStrLn $ "You entered: " ++ show radius ++ " area: " ++ show area ++ " volume: " ++ show volume

output:



Q10

Code:

main = do

    putStrLn $ "Enter side 1: "

    side1Str <-getLine

    putStrLn $ "Enter side 2: "

    side2Str <-getLine

    putStrLn $ "Enter side 3: "

    side3Str <-getLine

    let side1 = read side1Str::Double

    let side2 = read side2Str::Double

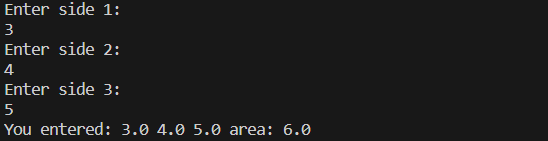
    let side3 = read side3Str::Double

    let s = (side1 + side2 + side3)/2

    let area = sqrt(s\*(s-side1)\*(s-side2)\*(s-side3))

    putStrLn $ "You entered: " ++ show side1 ++ " " ++ show side2 ++ " " ++ show side3 ++ " area: " ++ show area

Output:



Q11

Code:

main = do

    putStrLn $ "Enter a character: "

    charStr <- getLine

    let char = head charStr

    let upper = toEnum (fromEnum char - 32)

    putStrLn $ "You entered: " ++ [char] ++ ". The char in upper case is "++ [upper]

Output:



Q12

Code:

main :: IO ()

main = do

    putStrLn "Enter a character:"

    char <- getChar

    putStrLn $ "ASCII value: " ++ show (fromEnum char)

Output:



Q13

Code:

-- Function to calculate the sum of digits of a three-digit number

sumOfDigits :: Int -> Int

sumOfDigits num =

    let hundreds = num `div` 100            -- Extract the hundreds place digit

        tens = (num `div` 10) `mod` 10      -- Extract the tens place digit

        units = num `mod` 10                 -- Extract the units place digit

    in hundreds + tens + units               -- Return the sum of digits

-- Main function to test the sumOfDigits function

main :: IO ()

main = do

    putStrLn "Enter a three-digit number:"

    input <- getLine

    let number = read input :: Int

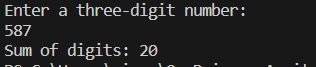
    let hundreds = number `div` 100

    let tens = (number `div` 10) `mod` 10

    let units = number `mod` 10

    putStrLn $ "Sum of digits: " ++ show (hundreds+tens+units)

Output:



Q14:

Code:

main = do

    let var1 = 10

    let var2 = 20

    putStrLn $ "Before swap the values of var1 and var2 are: " ++ show var1 ++ " " ++ show var2

    let var3 = var1

    let var1=var2

    let var2=var3

    putStrLn $ "After swap the values of var1 and var2 are: " ++ show var1 ++ " " ++ show var2

output:



Q15

Code:

main :: IO ()

main = do

    let var1 = 10

        var2 = 20

    putStrLn $ "Before swap the values of var1 and var2 are: " ++ show var1 ++ " " ++ show var2

    let (var1', var2') = (var2, var1)

    putStrLn $ "After swap the values of var1 and var2 are: " ++ show var1' ++ " " ++ show var2'

Output:



Q16

Code:

main = do

    putStrLn $ "Enter initial velocity: "

    input <- getLine

    let u = read input :: Double

    putStrLn $ "Enter acceleration: "

    input <- getLine

    let a = read input :: Double

    putStrLn $ "Enter time: "

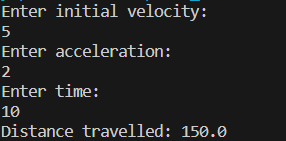
    input <- getLine

    let t = read input :: Double

    let s = (u\*t) + (0.5\*a\*t\*t)

    putStrLn $ "Distance travelled: " ++ show s

Output:



--------------------------------------------------------------------------------------------------------------------------------------

PART-B

Q1

Code:

main :: IO()

main = do

    print "Enter 2 numbers:"

    n1 <- getLine

    n2 <- getLine

    let num1 = read n1 :: Int

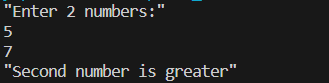
    let num2 = read n2 :: Int

    if num1>num2

        then print "First number is greater"

        else print "Second number is greater"

Output:



Q2

Code:

main :: IO()

main = do

    print "Enter 2 numbers:"

    n1 <- getLine

    n2 <- getLine

    let num1 = read n1 :: Int

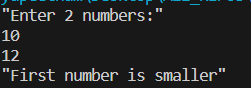
    let num2 = read n2 :: Int

    if num1<num2

        then print "First number is smaller"

        else print "Second number is smaller"

Output:



Q3

Code:

main :: IO()

main = do

    print "Enter 3 numbers:"

    n1 <- getLine

    n2 <- getLine

    n3 <- getLine

    let num1 = read n1 :: Int

    let num2 = read n2 :: Int

    let num3 = read n3 :: Int

    if (num1>num2) && (num2>num3)

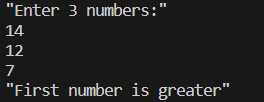
        then print "First number is greater"

    else if (num2>num3) && (num3>num1)

            then print "Second number is greater"

    else print "Third number is greater"

Output:



Q4

Code:

main = do

    putStrLn $ "Enter a number: "

    n <- getLine

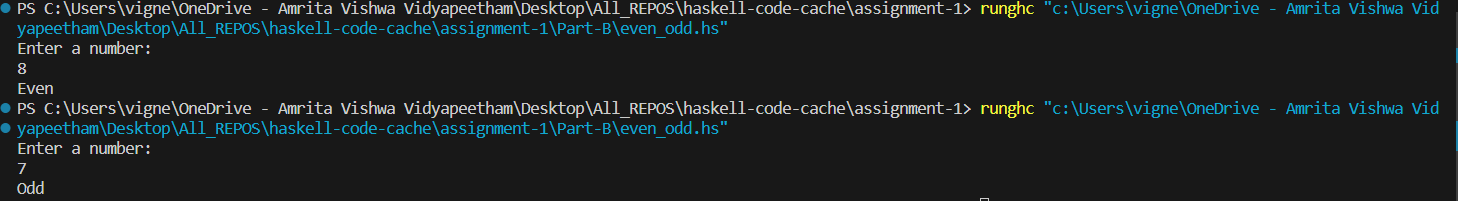
    let num = read n :: Int

    if (num `mod` 2) == 0

        then putStrLn $ "Even"

        else putStrLn $ "Odd"

Output:



Q5

Code:

Output: