Computer Programming Paradigms Lab – Lab 3

1. Write code in Haskell for inputting a number and finding its factorial.

```
-- Recursive function to calculate factorial
factorial :: Integer -> Integer
factorial 0 = 1
factorial n = n * factorial (n - 1)

main :: IO ()
main = do
    putStrLn "Enter a number to find its factorial:"
    input <- getLine
    let n = read input :: Integer
    if n >= 0
        then putStrLn $ "Factorial of " ++ show n ++ " is " ++ show
(factorial n)
        else putStrLn "Please enter a non-negative number!"
```

2. Write code in Haskell for inputting two positive numbers and multiplying them using repeated addition.
-- Function to multiply two numbers using repeated addition

```
multiply :: Int -> Int -> Int
multiply a 0 = 0
multiply a b = a + multiply a (b - 1)

main :: IO ()
main = do
    putStrLn "Enter the first number:"
    input1 <- getLine
    let a = read input1 :: Int

    putStrLn "Enter the second number:"
    input2 <- getLine
    let b = read input2 :: Int

    let result = multiply a b
    putStrLn $ show a ++ " multiplied by " ++ show b ++ " using repeated
addition is: " ++ show result
```

3. Define a Haskell function named "addUs" that adds 2 input numbers. Using this function as a building block, define a Haskell function "multiplyUs" that multiplies two input numbers. The multiplyUs function should cater to following:

i. Inputs may be siqned numbers for e.g. "multiplyUs (-2) * (3)" shorrld result in "-6" and "multiplyUs (-2) * (-6)" should result in “12”
j. It should use guarded expressions and recursion.
k. No need to write the main function to do user interaction. Writing definition for "addUs" and "multiplyUs" is sufficient. Explain type signature for your code.

```
-- Function to add two numbers
addUs :: Int -> Int -> Int
addUs x y = x + y

-- Function to multiply two numbers using repeated addition (and subtraction
for negative numbers)
multiplyUs :: Int -> Int -> Int
multiplyUs x 0 = 0
multiplyUs 0 y = 0
multiplyUs x y
    | x > 0 && y > 0 = addUs x (multiplyUs x (y - 1))
    | x < 0 && y > 0 = addUs x (multiplyUs x (y - 1))
    | x > 0 && y < 0 = addUs (-x) (multiplyUs x (y + 1))
    | x < 0 && y < 0 = addUs (-x) (multiplyUs x (y + 1))
```

4. Write code for the “hangman” game.

import System.IO

```
hangman :: IO ()
hangman = do
  putStrLn "Think of a word:"
  word <- sgetLine
  putStrLn "Try to guess it:"
  play word

sgetLine :: IO String
sgetLine = do
  x <- getCh
  if x == '\n'
    then do
      putChar x
      return []
    else do
      putChar '-'
      xs <- sgetLine
      return (x : xs)

getCh :: IO Char
getCh = do
  hSetEcho stdin False
  x <- getChar
  hSetEcho stdin True
  return x

play :: String -> IO ()
play word = do
  putStr "? "
  guess <- getLine
  if guess == word
    then putStrLn "You got it!!"
    else do
      putStrLn (match word guess)
      play word

match :: String -> String -> String
match xs ys =
  [ if elem x ys
    then x
    else '-'
  | x <- xs
  ]

main :: IO ()
main = hangman
```