Experiment No: 04

Aim: To design and develop Haskell code for given programming problems Part 01.

Lab Objective: Design and implement declarative programs in functional and logic programming languages.

Lab Outcomes: Design and Develop solution based on declarative programming paradigm using functional and logic programming (LO2)

Requirements: Any Text Editor and Glasgow Haskell Compiler 8.0+ Version

Performance: [Note: While writing the write up student need to change the wording such that it coveys that students have done all following steps. Also where ever output is generated the output must be written by the student]

Problem Statement 1.

Write a Haskell function to Implement safetail function that behaves in the same way as tail, except that safetail maps the empty list to the empty list, whereas tail gives an error in this case. Define safetail using: (a) a conditional expression; (b) guarded equations; (c) pattern matching.

[**Note:** Sample Code need to be executed and execution steps along with output must be recorded by the students]

Code Part 01(a)

```
safet :: [a] -> [a]
safet xs = if null xs then [] else (tail xs)
main :: IO [Char]
main = do
 putStrLn("Enter a string")
 name <- getLine
 return (safet name)
```

Code Part 01(b)

```
safet (x:xs)
 | null xs  = []
 | otherwise = xs
main :: IO [Char]
main = do
 putStrLn("Enter a string")
 name <- getLine
 return (safet [])
```

Code Part 01(c)

```
safet [] = []
safet (x:xs) = xs
main :: IO [Char]
main = do
 putStrLn("Enter a string")
 name <- getLine
 return (safet name)
```

Part 2. **Write Haskell function to recursive function** to multiply two natural numbers that uses predefined add function.

```
add :: Num a => a -> a -> a
add x y = x + y

multiply :: (Ord t, Eq t, Num t) => t -> t -> t
multiply x y
 | y == 0 = 0
 | y < 0 = (-1) * (add x (multiply x ((-y)-1)))
 | x < 0 = (-1) * (add (-x) (multiply (-x) (y-1)))
 | otherwise = (add x (multiply x (y-1)))
main = do
 return (multiply (-4) 5)
```

[Note: Update code to input two numbers interactively and convert to int and apply function.]

Part 3. **Write Haskell code** to represent infinite fibobacci series.

```
fibs = 0 : 1 : zipWith (+) fibs (tail fibs)
main :: IO [Int]
main = do
 return(take 10 fibs)
```

Part 4. **Write Haskell code using recursion** to find factorial of a number

```
factorial x
 | x == 0 = 1
```

```
 | x > 0 = x * factorial (x-1)
main = do
 putStrLn "\nEnter a positive integer: "
 m <- getLine
 let y = read m :: Int
 return(factorial y)
```

Conclusion: Thus we have understood how to create functional solution to programming problems using Haskell.

Reference:

1. Glasgow Haskell Project Home Page. https://www.haskell.org/
2. Learn You a Haskell for Great Good ! A Beginner's Guide http://learnyouahaskell.com/
3. Michael L Scott, 'Programming Language Pragmatics', 3rd Edition, Elsevier Publication.