MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS

CNG242 Programming Language Concepts – Spring 2021 – Lab 5: Haskell

1. Curried Functions

```
Prelude> max 4 5
5
Prelude> (max 4) 5
5
Prelude> add x y = x + y
Prelude> add 2 3
5
Prelude> newadd = add 5
Prelude> add 5 8
13
Prelude> newadd 8
13
Prelude> (\a -> rem a 10) 242
2
Prelude> lastDigit = (\a -> rem a 10)
Prelude> lastDigit 242
2
Prelude> lastDigitTwo = (`rem` 10)
Prelude> lastDigitTwo 15
5
Prelude> isGreater x y = if x>y then True else False
Prelude> isLessThanTen = isGreater 10
Prelude> isLessThanTen 20
False
Prelude> isLessThanTen 5
True
```

2. where

```
lambdaFunction x = x + (\y z -> (y+z)/2) 2 3

myWhereFunction x = x + secondfunction 2 3
                      where secondfunction y z = (y + z)/2

catordog x = “I love ” ++ identifyAnimal x
                    where identifyAnimal “cat” = “cats.”
                          identifyAnimal “dog” = “dogs.”
                          identifyAnimal _ = “all animals.”
```

```
*Main> lambdaFunction 8
10.5
*Main> myWhereFunction 8
10.5
*Main> catordog “cat”
"I love cats."
*Main> catordog “dog”
"I love dogs."
*Main> catordog “else”
"I love all animals."
```

3. let … in

```
lambdaFunction x = x + (\y z -> (y+z)/2) 2 3

myLetInFunction x = let myfunction y z = (y + z)/2
                    in x + myfunction 2 3

squareFunction k = let square x = x*x
                   in [square a  |  a<-k]
```

```
*Main> lambdaFunction 8
10.5
*Main> myLetInFunction 8
10.5
*Main> squareFunction [1,2,4,7]
[1,4,16,49]
```

4. Map

```
Prelude> :t map
map :: (a -> b) -> [a] -> [b]
Prelude> square a = a * a
Prelude> map square [1,2,3,4,5,6,7]
[1,4,9,16,25,36,49]
Prelude > map even [1,2,3,4,5,6]
[False,True,False,True,False,True]
```

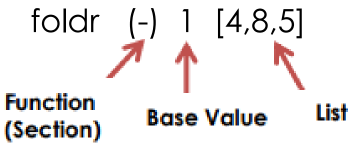
5. Filter

```
Prelude> :t filter
filter :: (a -> Bool) -> [a] -> [a]
Prelude> [a  |  a<-[1,2,3,4,5,6,7],even a]
[2,4,6]
Prelude > filter even [1,2,3,4,5,6,7]
[2,4,6]
```

6. zipWith

```
Prelude > :t zipWith
zipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
Prelude> zipWith (++) ["Hello","CNG"] ["World","242"]
["HelloWorld","CNG242"]
Prelude> zipWith max [1,5..17] [6,8..14]
[6,8,10,13,17]
```

7. foldr & foldl



```
foldr (-) 1 [4,8,5]
 4 – (foldr (-) 1 [8,5])
 4 – (8 – (foldr (-) 1 [5]))
 4 – (8 – (5 – (foldr (-) 1 [])))
 4 – (8 – (5 – 1))
 4 – (8 - 4)
 4 – 4 = 0
```

```
foldl (-) 1 [4,8,5]
foldl (-) (1 - 4) [8,5]
foldl (-) ((1 – 4) - 8) [5]
foldl (-) (((1 - 4) - 8) - 5) []
((1 - 4) - 8) – 5
((-3) - 8) – 5
(-11) – 5
-16
```

Practical Exercises (Part 1):

***Please attempt to the exercises in part one by following the material of the lab sessions we considered in the previous weeks.**

1. Implement a function which takes a list and returns the number of numeric characters in the list. Try to solve this question in two different ways, one with recursion and one with list comprehension.

   Sample run:

```
*Main> countNumbersRecursive "Hello world 123"
3
*Main> countNumbersListComp  "Hello world 123"
3
*Main> countNumbersRecursive "More than 100 students are taking CNG 242 now"
6
```

2. Implement a function which takes a j value and calculate the result of the following equation;

$$\sum_{i=0}^{j} \frac{i^2 + 6}{2i + 1}$$

   Sample Run:

```
*Main> sumEquation 10
38.38046611111626
```

Practical Exercises (Part 2):

***Implement and/or try these exercises with anything that we have covered in all labs. Try to use something from this worksheet while solving part 2 exercises.**

1. The implementations of the letInFunction, lambdaQuestion and lambdaQuestion functions can be found below. You need to trace the following functions and to provide the output of them for the following Haskell function calls.

```
letInFunction = let a = 1
                    f x = a + (g x)
                    g x = x + 2
                    in f 2 + let a = 4
                             g x = (x + 1)
                             in (f 3)

mapQuestion xs = map f xs where f x = x * 2 + 3

lambdaQuestion xs = foldr (\x y -> x + y) 1 xs
```

Function Call	Output
letInFunction	
mapQuestion [1,2,3]	
lambdaQuestion [1,2,3]	

2. Implement the set union, the set intersect and set difference functions **using higher order functions.** You can also try to implement a function that takes union of two lists without the intersection (rest).

Sample Runs:

```
*Main> setUnion [1,2,3] [3,4]
[1,2,3,4]
*Main> setIntersection [1,2,3] [3,4]
[3]
*Main> setDifference [1,2,3] [3,4]
[1,2]
*Main> setRest [1,2,3] [3,4]
[1,2,4]
```

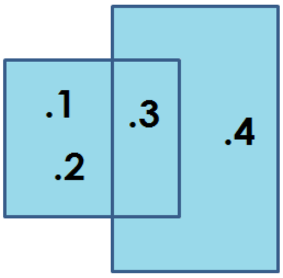


Fig 1. Union Example

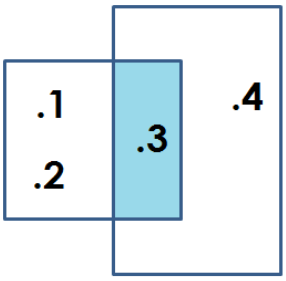


Fig 2. Intersection Example

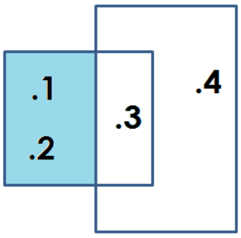


Fig 3. Difference Example

References:

1. Learn You a Haskell <http://learnyouahaskell.com/chapters>
2. A Gentle Introduction to Haskell <http://www.haskell.org/tutorial/index.html>
3. H-99: Ninety-Nine Haskell Problems <https://wiki.haskell.org/H-99:_Ninety-Nine_Haskell_Problems>