MIDDLE EAST TECHNICAL UNIVERSITY, NORTHERN CYPRUS CAMPUS

CNG242 Programming Language Concepts – Lab 2: Haskell

1. Ranges

```
Prelude> [1..5]
[1,2,3,4,5]
Prelude> [1,3..11]
[1,3,5,7,9,11]
Prelude> ['a'..'z']
"abcdefghijklmnopqrstuvwxyz"
Prelude> ['A'..'Z']
"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
Prelude> take 5 [1..]          <-----   Lazy Evaluation
[1,2,3,4,5]
Prelude> take 3 (repeat 5)
[5,5,5]
Prelude> take 5 (cycle [1,2,3])
[1,2,3,1,2]
```

2. List Comprehensions

```
Prelude> ~~let~~ xs = [1,2,3,4,5,6,7,8,9,10]
Prelude> [x | x<-xs, even x]
[2,4,6,8,10]

Prelude> [if x<5 then "Hello" else "Hi" | x<-xs, even x]
["Hello","Hello","Hi","Hi","Hi"]

Prelude> ~~let~~ removeUpperCase cs = [c| c<-cs, not(c `elem` ['A'..'Z'])]
Prelude> removeUpperCase "Computer Engineering"
"omputer ngineering"
```

3. Functions with different patterns

```
----------------firstHaskell.hs--------------------
myAge 1 = "I am 1 year old"
myAge 2 = "I am 2 years old"
myAge 3 = "I am 3 years old"
myAge _ = "I am older than 3 years old"
---------------------------------------------------
*Main> myAge 1
"I am 1 year old"
*Main> myAge 3
"I am 3 years old"
*Main> myAge 5
"I am older than 3 years old"
```

4. Recursive Functions

Factorial

4! = 4 * 3 * 2 * 1 = 24

```
----------------firstHaskell.hs--------------------
factorial 0 = 1
factorial n = n * factorial (n-1)
---------------------------------------------------
*Main> factorial 4
24
```

$$\begin{tikzcd}
4 \ast \text{factorial } 3 \arrow[dr] &&&&&&&& 4 \ast 6 = \boxed{24} \\
& 3 \ast \text{factorial } 2 \arrow[dr] &&&&&& 3 \ast 2 = 6 \arrow[ur] & \\
&& 2 \ast \text{factorial } 1 \arrow[dr] &&&& 2 \ast 1 = 2 \arrow[ur] && \\
&&& 1 \ast \text{factorial } 0 \arrow[dr] && 1 \ast 1 = 1 \arrow[ur] &&& \\
&&&& 1 \arrow[ur] &&&&
\end{tikzcd}$$

Re-implementation length function using a recursive function

```
----------------firstHaskell.hs--------------------
myLen [] = 0
myLen (x:xs) = 1 + myLen xs
---------------------------------------------------
*Main> myLen [1,2,3]
3
```

$$\begin{tikzcd}
1 + \text{myLen } [2, 3] \arrow[dr] &&&&&& 1 + 2 = \boxed{3} \\
& 1 + \text{myLen } [3] \arrow[dr] &&&& 1 + 1 = 2 \arrow[ur] & \\
&& 1 + \text{myLen } [\,] \arrow[dr] && 1 + 0 = 1 \arrow[ur] && \\
&&& 0 \arrow[ur] &&&
\end{tikzcd}$$

5. Data Types

a. Basic Data Type

```
Prelude> :type 'a'
'a' :: Char
Prelude> :type "CNG"
"CNG" :: [Char]
Prelude> :type True
True :: Bool
Prelude> :type 242
242 :: Num a => a
Prelude> :type [1,2,3]
[1,2,3] :: Num t => [t]
```

b. Function Type

```
*Main> :t factorial
factorial :: (Eq a, Num a) => a -> a
```

c. Dealing with numbers: An example

`fromIntegral` function takes an integral number and turns it into a more general number.

```
Prelude> length [1,2,3,4,5] + 3.2

<interactive>:21:22:
    No instance for (Fractional Int) arising from the literal `3.2'
    Possible fix: add an instance declaration for (Fractional Int)
    In the second argument of `(+)', namely `3.2'
    In the expression: length [1, 2, 3, 4, ....] + 3.2
    In an equation for `it': it = length [1, 2, 3, ....] + 3.2

Prelude> :t length [1,2,3,4,5]
length [1,2,3,4,5] :: Int
Prelude> fromIntegral(length [1,2,3,4,5]) + 3.2
8.2
Prelude> :t fromIntegral(length [1,2,3,4,5])
fromIntegral(length [1,2,3,4,5]) :: Num b => b
```

d. New Type Definition (Disjoint Union)

```
data SchoolMember = Student [Char] [Char] Integer | Teacher [Char]
[Char] [Char] deriving (Show, Eq, Ord)
```

e. Implementation of `toString` function for the `SchoolMember` data type

```
toString (Student name sname id) = "I am a student. My name is " ++ name
++ " " ++ sname ++ ". My student ID is " ++ show id

toString (Teacher name sname dept) = "I am a teacher. My name is " ++
name ++ " " ++ sname ++ ". I belong to " ++ dept
```

Sample Run:

```
Ok, modules loaded: Main.
*Main> let myTeacher = Teacher "Enver" "Ever" "CNG"
*Main> toString myTeacher
"I am a teacher. My name is Enver Ever. I belong to CNG"
```

Practical Exercises:

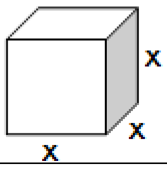
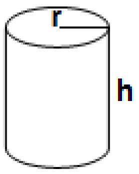
a. Implement the power function where power x y evaluate to x^y

b. Write a function that takes a string and returns the number of vowels in the string

c. Implement f(x) by using patterns. (See Item 3)

$$f(x) = \begin{cases} 1 - x, & x = 5 \\ \frac{x}{x^2}, & x = 10 \\ x, & Otherwise \end{cases}$$

d. Define a new type for `ThreeDShapes`. It should have a data constructor for `Cube` and `Cylinder`. `Cube` should have side length and `Cylinder` should have radius and height. You need to implement two functions related to this type.
   - `volume` function calculates the volume of the 3D shape
   - `surfaceArea` function calculates the surface area of the 3D shape.

Volume and Surface Area Formulas for Cube and Cylinder		
Cube		Volume = x^3 Surface Area = $6x^2$
Cylinder		Volume = $\pi r^2 h$ Surface Area = $2\ \pi rh + 2\pi r^2$

e. Write a function that takes two lists where one list contains letter grades and another list contains course credits. Your function should calculate and then return the GPA (Grade Point Average). The weight of the letter grades are as follows: A = 4, B = 3, C = 2, D= 1 and F = 0.

Sample Run:

```
calculateGPA ['A', 'B', 'A', 'C'] [4, 2, 3, 3]
3.33
```

References

Miran Lipovača, *Learn You a Haskell for Great Good! A beginner's guide to Haskell*, No Starch Press, Daly City, California, United States, 2011

Useful Links

Learn You a Haskell <http://learnyouahaskell.com/chapters>