St. Francis Institute of Technology
Borivli (West), Mumbai-400103
Department of Information Technology

Experiment – 5

1. Aim: To implement list comprehensions using Haskell Programming

2. Objective: After performing the experiment, the students will be able to –

- Create and display lists□
- Perform list comprehensions operations□
- Develop small user defined functions using Haskell programming□

3. Lab objective mapped: To **understand** and **implement** declarative programming paradigm through functional programming (PSO2) (PO1)

4. Prerequisite: Knowledge of arithmetic operations, logical and comparative operations

5. Requirements: The following are the requirements –

- Haskell Compiler□

6. Pre-Experiment Theory:

List in Haskell is very important

- It's the most used data structure□
- In Haskell, lists are a **homogenous** data structure. It stores several elements of the same type.□
- That means that we can have a list of integers or a list of characters but we can't have a list that has a few integers and then a few characters.□

List in Haskell are denoted by square brackets and the values in the list are separated by commas. Examples

```
Prelude> let x1=[1,2,3,4,5,6]
```

Is a list consisting of Num type of data. The list is stored in variable x1.

```
Prelude>let x2=[1.1,2.1,3.1,4.1,5.1,6.1]
```

Is a list consisting of fractional type of data. The list is stored in variable x2.

```
Prelude>let x3=[‘a’,’b’,’c’]
```

Is a list consisting of character type of data. The list is stored in variable x3.

```
Prelude>let y=[1,2,3,’c’, ‘a’]
```

The above example will give error as list y consist of heterogeneous data types.

The List comprehension commands are as under-

- **head**- Takes a list and returns its head. The head is the first element of the list.
- **tail** -Takes a list and returns its tail. The tail in list consists of the remaining elements of list
- **last**- Takes a list and returns its last element.
- **init**- Takes a list and returns everything except its last element.
- **length**- Takes a list and returns its length.
- **null**- Checks if the list is a null list. If the list is a null list then it returns true else false.
- **reverse**-Reverses a given list
- **take**- Takes a number and a list. It extracts that many elements from a list.
- **drop**- Takes a number and a list. It drops that many numbers from the beginning of a list.
- **maximum**- Takes a list and returns the maximum value of the list.
- **minimum**- Takes a list and returns the minimum value of the list.
- **sum**- Takes a list and evaluates the sum of all numbers of a list.
- **product**- Takes a list and evaluates the product of all numbers of a list.
- **elem**-Takes a thing and a list of things and tells us if that thing is an element of list

7. Laboratory Exercise

A. Steps to be implemented

To work on Prelude prompt, use the command ***ghci.*** To work on Haskel files, follow the instructions given below

1. Open notepad/text editor
2. Write your function code and save the file as ‘filename.hs’ file.
3. While saving make sure to keep *save as type* as ‘All files’
4. Open command prompt and type ‘ghci’.This will take you to ‘Prelude’ prompt.
5. Change the path (if required) to the place where you have saved your file using the following command **prelude>: cd** ‘write your complete path here’
6. To compile, load the program using the following command **prelude>: l** your-file name [do not write the file name with .hs extension]
7. To run, write the name of the function at prelude prompt **Main>: name of function**
8. Check the output

B. Program Code

1. Execute the following inbuilt functions using Haskell compiler at prelude prompt
   - Prelude> x=[1,2,3]
   - Prelude> print x
   - Prelude> let x1=['a','b']
   - Prelude> print x1
   - Prelude> [1,2,3,4]++[5,6,7]
   - Prelude> [1.1,2.1,3.1,4.1]++[5.1,6.1,7.1]
   - Prelude>['a','b']++['s','v']
   - Prelude> "hello"++" "++"world"
   - Prelude> 'A': "Small Dog"
   - Prelude> 'A':" "++"Small Dog"
   - Prelude> '1':" "++"Small Dog"

- Prelude> 6:[1,2,3,4]
- Prelude> 6.1:[1,2,3,4]
- Prelude> 6.1:[1,2,3,4]++ 4:[1,2,3,4]
- Prelude> []
- Prelude> "Steve Buscemi" !!4
- Prelude> let x="Steve Buscemi" !!4
- Prelude> print x
- Prelude> let b=[[1,2,3],[4,5,6],[7,8,9]]
- Prelude> print b
- Prelude> b ++[[1,1,1]]
- Prelude> let y1=b++[[1,1,1]]
- Prelude> print y
- Prelude>length [4,5,6,8]
- Prelude> length []
- Prelude> length [[]]
- Prelude> length [[[]]]
- Prelude> length [[],[]]
- Prelude> reverse [7,5,4,9,10]
- Prelude> take 3 [7,5,4,9,10]
- Prelude> drop 3 [7,5,4,9,10]
- Prelude>head [7,5,4,9,10]
- Prelude> tail [7,5,4,9,10]
- Prelude> init[7,5,4,9,10]
- Prelude> last [7,5,4,9,10]
- Prelude> sum[7,5,4,9,10]
- Prelude> product[7,5,4,9,10]
- Prelude> maximum[4,6,7,8,9]
- Prelude> minimum[2,4,5,6,7]
- Prelude> 4 `elem` [6,7,8,4]
- Prelude> zipWith(+)[1,2,3] [4,5,6]
- Prelude> zipWith(-)[1,2,3] [4,5,6]
- Prelude> zipWith(*)[1,2,3] [4,5,6]
- Prelude> zipWith(/)[1,2,3] [4,5,6]

2. Perform the following list comprehensions using Haskell compiler at the prelude prompt and explain each comprehension in your own words
   - Prelude>[x*2 | x <- [1..10]]
   - Prelude> [x*2 | x <- [1..100], x ‘mod’ 2 == 0]
   - Prelude> [x | x <- [10..20], x /= 13, x /= 15, x /= 17]
   - Prelude> [x | x <- [1..100], x ‘mod’ 7 == 0, x >=50]
   - Prelude> [x^2 | x <- [1..10]]
   - Prelude> [2^x|x<-[1..20]]
   - Prelude> let nouns=["hobo","frog","pope"]
   - Prelude> let adj=["lazy", "scram","grouchy"]
   - Prelude> [adj++" "++noun|adj<-adj,nouns<-nouns]
   - Prelude> fst(8,9)
   - Prelude> snd(8,0)
   - Prelude> snd(8,"wow")

- Prelude> zip [1,2,3,4] [3,3,3,3]
- Prelude> zip [1..5] [3,3,3,3]
- Prelude> zip [1..5] ["a","b","c","d","e"]

3. Write a function in Haskell to

- Define a function doublePost that doubles the positive elements in the list of integers
- Define a function **spaces n** which returns a string of n spaces
- Generate a list of **triples (x,y,z)** such that $\mathbf{x^2+y^2=z^2}$
- Define a function **factor n** which returns a list of integers that divide n. Omit the trival factors of 1 and n

9. Post Experimental Exercise-

A. Questions:

1. List important characteristics of Lists in Haskell
2. Write a Haskell snippet to create a list of first 10 numbers in numbers
3. Write a Haskell snippet to filter the elements from the list whose square root is greater than seven
4. Write a Haskell snippet to implement Fibonacci series. Define an expression fibs :: [Integer] that generates this infinite sequence.

B. Results/Observations/Program output:

Present the program input/output results if any and comment on the same.

C. Conclusion:

1. Write what was performed in the experiment
2. Write which tools you used to perform the experiment
3. Write what you inferred from the output obtained

D. References:

[1] https://www.haskell.org/

[2] http://learnyouahaskell.com/

[3] Michael L Scott, " Programming Language Pragmatics", Third edition, Elsevier publication