Experiment No: 03

Aim: Tutorial on lists in Haskell.

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

Theory: **Predefined Types and Classes**

The Haskell Prelude contains predefined classes, types, and functions that are implicitly imported into every Haskell program.

Standard Haskell Types : These types are defined by the Haskell Prelude.

1. **Numeric types**: Haskell provides several kinds of numbers; the numeric types and the operations upon them have been heavily influenced by Common Lisp and Scheme. Numeric function names and operators are usually overloaded, using several type classes with an inclusion relation.

   The class Num of numeric types is a subclass of Eq, since all numbers may be compared for equality; its subclass Real is also a subclass of Ord, since the other comparison operations apply to all but complex numbers.

   The class Integral contains integers of both limited and unlimited range; the class Fractional contains all non-integral types; and the class Floating contains all floating-point types, both real and complex.

   The Prelude defines only the most basic numeric types: fixed sized integers (**Int**), arbitrary precision integers (**Integer**), single precision floating (**Float**), and double precision floating (**Double**). Other numeric types such as rationals and complex numbers are defined in libraries.

2. **Boolean**: Bool is an enumeration. The basic boolean functions are && (and), || (or), and not. The name otherwise is defined as True to make guarded expressions more readable. Constructor for Bool is :
   data Bool = False | True deriving (Read, Show, Eq, Ord, Enum, Bounded)
3. **Characters and Strings** : The character type Char is an enumeration

whose values represent Unicode characters. Type Char is an instance of the classes Read, Show, Eq, Ord, Eum, and Bounded. A string is a list of characters: type String = [Char]
For example, "A string" abbreviates ['A',' ' ,'s','t','r','i','n','g']

4. **Lists:** Lists are an algebraic datatype of two constructors, although with special syntax, as described below. The first constructor is the null list, written `[]' ("nil"), and the second is `:' ("cons").
Data [a] = [] | a : [a] deriving (Eq, Ord)
The module PreludeList defines many standard list functions. Arithmetic sequences and list comprehensions, are two convenient syntax for special kinds of lists are available. Lists are an instance of classes Read, Show, Eq, Ord, Monad, Functor, and MonadPlus.

5. **Tuples:** Tuples are algebraic data types. Each tuple type has a single constructor. All tuples are instances of Eq, Ord, Bounded, Read, and Show (provided, of course, that all their component types are as well). There is no upper bound on the size of a tuple, but some Haskell implementations may restrict the size of tuples, and limit the instances associated with larger tuples.
However, every Haskell implementation must support tuples up to size 15, together with the instances for Eq, Ord, Bounded, Read, and Show.
fst, snd, curry, and uncurry functions are defined for pairs (2-tuples). Similar functions are not predefined for larger tuples.

6. **The Unit Data type** : The unit datatype ( ) has one non-_|_ member, the nullary constructor ().
data () =() deriving (Eq, Ord, Bounded, Enum, Read, Show)

7. The **IO** and **IOError** Types: The **IO type** serves as a tag for operations (actions) that interact with the outside world. The IO type is abstract i.e. no constructors are visible to the user. IO is an instance of the Monad and Functor classes.
IOError is an abstract type representing errors raised by I/O operations. It is an instance of Show and Eq. Values of this type are constructed by the various I/O functions. The Prelude contains a few I/O functions like print, getChar, getLine, readLn etc.

8. **Other Types**: Consider following constructor
data Maybe a = Nothing | Just a deriving (Eq, Ord, Read, Show)
data Either a b = Left a | Right b deriving (Eq, Ord, Read, Show)
data Ordering =LT | EQ | GT deriving
(Eq,Ord,Bounded,Enum,Read,Show)

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]

Students need to perform following steps to complete this Experiment:

1. Understand how to define different lists and tuples in ghci and note their type
2. Understand various Prelude functions that operate on list, their working and their type signatures
3. Understand how to create ranges and concept of infinite list.
4. Understand what is meant by List comprehention

Part 1. Declare lists and typles in ghci

1. To begin, we will create list of Int data of size 4:

```
Prelude> let a = [2,3,4,5]
Prelide> a <enter>
Prelude> [2, 3, 4, 5]
```

2. Next create a list of 5 Float numbers with name x and print its values. Write its code and output
3. Next create a list of characters in your name and store with name y and print its values. Write its code and output
4. Now Create a list of 3 Strings with name **myname** that contain your first name, middle name and last name and print its values. Write its code and output.
5. Execute these command and write down output (Learn use of ++, : and !! operators (concatenation, cons, index operator))
   - `Prelude>[1,2,3,4] ++ [9,10,11,12]`
   - `Prelude> "hello"++" "++"world!"`
   - `Prelude> [‘w’,’h’] ++ [‘a’,’t’]`
   - `Prelude>'A':" SMALL CAT"`
   - `Prelude>1:[4,6]`
   - `Prelude>1:4:6:[]`
   - `Prelude>1:4:6`
   - `Prelude > [9.4,33.2,96.2,11.2,23.25]!!3`
   - `Prelude > [9.4,33.2,96.2,11.2,23.25]!!0`
   - `Prelude > [9.4,33.2,96.2,11.2,23.25]!!5`
   - `Prelude> let x=[1,2,3]; y=[5,6]`

- Prelude> x
- Prelude> y
- Prelude> let z=[x,y]
- Prelude> z
- Prelude> :type x
- Prelude>:type y
- Prelude>:type z

Part 2. Understand Various Prelude functions operating on lists

1. Students need to fill following table to complete this step.

Sr.	Name	Example	Output	Type signature
1	head	head [4,2,5,6]	4	head :: [a] -> a
2	tail	tail [5,4,3,2,1]	[4,3,2,1]	
3	last	last [5,4,3,2,1]	1	
4	init	init [4,2,5,6]	[4,2,5]	
5	length	length “hi there”	8	
6	null	null []	True	
7	reverse	reverse [5,6,3]	[3,6,5]	
8	take	take 3 [5,6,2,3,4,3]	[5,6,2]	
9	drop	drop 2 [4,5,2,2,4]	[2,2,4]	
10	elem	4 `elem` [3,4,5,6]	True	
11	maximum	minimum [8,4,2,1,5]	1	
12	minimum	maximum [1,9,2,3,4]	9	

Part 3. Understand how to create Range and infinite lists
Execute following commands and and note output:

1. Prelude> let alphabet=[‘a’..’z’]
   Prelude> alphabet
2. Write range of number to create list of first 20 even numbers
3. Write range of number to create list of multiple of 4 starting at 8 up to 45
4. Prelude> take 24 [13,26..]
5. Prelude> take 10 (cycle [1,2,3])
6. Prelude> take10 (repeat 5)

Part 4. Understand List Comprehensions

Complete following table by executing every command and noting the elements in output/created list.

sr.	List Comprehension	Elements of list
1	[x^2 | x <-[1..5]]	
2	[x*2 | x<-[6..12],x*2<=15]	
3	[x|x←[50..100], x `mod` 7==3]	
4	[x|x<-[10..20],x/=13,x/=15,x/=19]	
5	[x*y|x←[2,5,10], y←[8,10,11], x*y>50]	
6	[x+y|x<-[2,5,10],y<-[8,10,11]]	
7	[(a,b,c) |c<-[1..10],b<-[1..c],a<-[1..b],a^2+b^2==c^2]	

Exersize: Write answer to following questions :

a) Can we create infinite lists in Haskell? Which concept in Haskell allows us to define and use infinite lists.
b) Can we create range with negative command difference? If possible create list of multiples of 2 from 20 to 0 using Haskell statement.
c) Explain what type of tuples (a,b,c) are created using Part 4 row 7 statement.
d) Explain concept of function currying.

Conclusion: Thus we have learned about basic data types in Haskell and learned how to work with Haskell Lists.

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.