Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Laboratory 9

Title of the Laboratory Exercise: Higher order functions and Folding

1. Introduction and Purpose of Experiment

   Students apply higher order programming to solve problems using functional programming paradigm

2. Aim and Objectives

   Aim

   To apply higher order programming to solve problems using functional programming paradigm

   Objectives

   At the end of this lab, the student will be able to

   - Apply higher order programming for solving problems
   - Express functions in terms of other functions

3. Experimental Procedure

For the problems listed below, design the data structures, algorithm(s) and write the program(s). Tabulate the output for various inputs and verify against expected values. Compare the programming method in Haskell with C programming language. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

a. Find if a given list is palindrome using strings and higher order functions in Haskell

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Documentation:	
a. Procedure and Algorithm(s):	**Procedure:** 1. Creating a new Haskell File Haskell source files are ASCII based text file with a `.hs` extension. The execution of the Haskell Project usually begins from `Main.hs` and hence we name the file Main.hs, it can be anything if its going to a simple project, for the sake of organization the file that contains the driver function or main function is inside the Main.hs. 2. Design the functions required Since Haskell is a functional programming language, the required functions need to be decomposed into sub functions, the operations that needs to be done in the function also needs to be purely functional, and side-effects needs to be avoided as much as possible. 3. Documentation Write documentation for the functions that are used in the source file along with its input and outputs, write the function signatures to enforce typing in Haskell. 4. Running and testing The functions written can be individually tested in GHCI, which is an interactive version of the Glasgow Haskell Compiler, this will us know test atomic functions, which are a part of more functions, and make it easier to debug the programs, since there are no side-effects testing becomes easier. Load the `.hs` file into GHCI by typing `:l Main.hs` , this will compile and load the function written in the file and each of the function can be called specifically by just typing the name of the function, to see the type of function `:t` can be used. **Algorithms:**

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

	isPalindrome
	Params: a list of elements lst
	`Step 1: Start`
	`Step 2: return lst == reverse lst`
	`Step 3: Stop`
	
	reverse
	Params: a list of elements lst
	`Step 1: Start`
	`Step 2: Take every last element of list and push it to the front of the list recursively`
	`Step 3: Stop`

b. Conclusions :	Higher Order programming is an extremely useful construct that can shorten programs and make them more manageable. Common patterns detected in multiple functions can be rewritten efficiently using higher order programming. The knowledge of inbuilt functions such as foldr, foldr, map, reduce, filter, all, or, etc. was also obtained. HUnit testing can be used by software developers to test and debug Haskell code, since we have already done functional decomposition it is easy to debug such functions.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Results and Discussions:

Screenshot:

```
reverse' :: [a] -> [a]
-- reverse' = foldl (flip (:)) []
reverse' = foldl (\rest last -> last : rest) []

isPalindrome :: Eq a => [a] -> Bool
isPalindrome x = reverse' x == x
```

OUTPUT :

```
*Main> isPalindrome "palindrome"
False
*Main> isPalindrome "steponnopets"
True
*Main> isPalindrome "12321"
True
*Main> isPalindrome "kayak"
True
*Main> isPalindrome "satyajit"
False
*Main> isPalindrome [4,3,4]
True
*Main> isPalindrome [True,False,True]
True
*Main> isPalindrome ["satyajit","ghana","satyajit"]
True
*Main>
```

Discussion:

A palindrome is a string, or a list, which is same as it's reverse. Using this exact definition, we define the palindrome in our program, i.e. a function that takes a list as an argument, the constraint is that the elements in the list are comparable to Equals, hence the type constraint `Eq`. The list is reversed and compared to the actual list, if they are same then it is a palindrome else not. isPalindrome is a function that takes in a list as the argument and returns a Boolean, either True or False.

The reverse function uses Higher order function `foldl` to reverse the list. Let's see how this works out. Foldl folds the list from the left towards the right, at every recursive step it takes in the last element and the rest of the list, we need to make the last element the first element and recursively do the same thing to the rest

of the list, this does last : last-1: last -2: . . . : first, which is the reverse of the list, the base case of this is []. i.e. if no argument are passed to it then it's an empty list.

Taking the example of the string "madam" passed to the isPalindrome function, it is first reversed, using foldl Which makes it

```
('m':('a':('d':('a':('m':[])))))
"madam"
```

This is then compared to the original parameter "madam", since both of them are equal True is returned and it is a palindrome.