

20CYS312 – PPL –LAB EXERCISE 4

Name: Kiruthik Pranav P V

Date:20.12.2024

Roll No:CH.EN.U4CYS22026

Github Link: <https://github.com/kpsan12/Haskell-Programming/tree/main/lab4>

1) Implement a function swapTuple that takes a tuple (a, b) and swaps its elements, i.e., returns the tuple (b, a).

Objective:

To Implement a function **swaptuple** that takes a tuple (a, b) and swaps its elements, i.e., returns the tuple (b, a).

Program Code:



The image shows a terminal window with a nano text editor. The title bar indicates the user is 'asecomputerlab@asecomputerlab' in the directory '~/Documents'. The menu bar includes 'File', 'Edit', 'View', 'Search', 'Terminal', 'Tabs', and 'Help'. There are three tabs open, all with the same title 'asecomputerlab@aseco...'. The current file is 'qn1.hs'. The code inside the editor is as follows:

```
swaptuple :: (a, b) -> (b, a)
swaptuple (a, b) = (b, a)
```

Explanation Of the Code:

- In the first line, we define the type signature: `swaptuple :: (Int, Int) -> (Int, Int)`, which states that the input and output are tuples of two integers.
- In the second line, `swaptuple (x, y) = (y, x)`, we assign `x` to the first element and `y` to the second element of the input tuple, then return the tuple with swapped elements `(y, x)`.

Input/Output Examples:

Input: `("Hi",5)`Output:`(5,"Hi")`

Screenshot:

A screenshot of a terminal window. The title bar shows the user 'asecomputerlab' and the directory '~/Documents/franav/lab'. The terminal has a menu bar with 'File', 'Edit', 'View', 'Search', 'Terminal', 'Tabs', and 'Help'. Below the menu bar, there are three tabs, all labeled 'asecomputerlab@aseco...'. The main terminal area shows a prompt 'asecomputerlab@asecomputerlab:~/Documents/franav/lab4\$' followed by the command 'ghci'. The output shows 'GHCi, version 8.0.2: http://www.haskell.org/ghc/ :? for help'. The user enters 'Prelude> :l qn1.hs', and the output shows '[1 of 1] Compiling Main (qn1.hs, interpreted)' and 'Ok, modules loaded: Main.'. The user then enters '*Main> swaptuple("hello",1)', and the output shows '(1,"hello")'. Finally, the user enters '*Main>' and the prompt is ready for the next command.

```
asecomputerlab@asecomputerlab: ~/Documents/franav/lab
File Edit View Search Terminal Tabs Help
asecomputerlab@aseco... x asecomputerlab@aseco... x asecomputerlab
asecomputerlab@asecomputerlab:~/Documents/franav/lab4$ ghci
GHCi, version 8.0.2: http://www.haskell.org/ghc/ :? for help
Prelude> :l qn1.hs
[1 of 1] Compiling Main (qn1.hs, interpreted)
Ok, modules loaded: Main.
*Main> swaptuple("hello",1)
(1,"hello")
*Main> 
```

Conclusion:

The swapTuple function takes a tuple of two integers, swaps their positions, and returns the swapped tuple as the output. It effectively uses pattern matching to manipulate tuple elements in Haskell.

2) Write a function multiplyElements that takes a list of numbers and a multiplier n, and returns a new list where each element is multiplied by n. Use a list comprehension for this task.

Objective:

To Write a function `multiplyElements` that takes a list of numbers and a multiplier n, and returns a new list where each element is multiplied by n. Use a list comprehension for this task.

Program Code:

A screenshot of a terminal window with a dark background. The terminal title bar shows 'asecomputerlab@asecomputerlab: ~'. The menu bar includes 'File Edit View Search Terminal Tabs Help'. There are two tabs: 'asecomputerlab@asecomputerlab: ~/Documents/franav/lab4' and 'asecomputerlab@asecomputerlab: ~/Documents/franav/lab4'. The terminal text shows 'GNU nano 2.9.3' at the top right. The code being edited is:

```
multiplyElements :: [Int] -> Int -> [Int]
multiplyElements xs n = [x * n | x <- xs]
```

Explanation Of the Code:

- The type signature defines that the function takes a list of integers and a multiplier as inputs and returns a new list of integers.
- The list comprehension iterates through each element of the input list, multiplies it by the multiplier, and constructs a new list with the resulting values.

Input /Output Examples:

Input: [1,2,3,4]2 Output:[2,4,6,8]

Input:[1,2,3,4]10 Output:[10,20,30,40]

Screenshot:

```
*Main> :l qn2.hs
[1 of 1] Compiling Main                ( qn2.hs, interpreted )
Ok, modules loaded: Main.
*Main> multiplyElements[1,2,3,4]2
[2,4,6,8]
*Main> multiplyElements[1,2,3,4]6
[6,12,18,24]
*Main> multiplyElements[1,2,3,4]10
[10,20,30,40]
*Main> 
```

Conclusion:

The multiplyElements function efficiently generates a new list by applying the multiplier to every element of the input list, using Haskell's concise list comprehension syntax.

3) Write a function filterEven that filters out all even numbers from a list of integers using the filter function.

Objective:

To Write a function filterEven that filters out all even numbers from a list of integers using the filter function.

Program Code:

```
asecomputerlab@asecomputerlab: ~/Documents/franav/lab4  ×  asecomputerlab@asecomputerlab: ~/Docum
GNU nano 2.9.3 qn3.hs
filterEven :: [Int] -> [Int]
filterEven xs = filter odd xs
```

Code Explanation:

- The type signature specifies that the function takes a list of integers as input and returns a list of integers.
- The filter function is used to retain only the elements that satisfy a condition. Here, the condition is odd, which checks if a number is not divisible by 2, effectively removing all even numbers

Input / Output Examples:

Input:-[1,2,3,4,5] Output:[1,3,5]

Input:[1,2,3,4,5,6,7] Output:[1,3,5,7]

Screenshot:

```
*Main> :l qn3.hs
[1 of 1] Compiling Main                ( qn3.hs, interpreted )
Ok, modules loaded: Main.
*Main> filterEven[1,2,3,4,5]
[1,3,5]
*Main> filterEven[1,2,3,4,5,6]
[1,3,5]
*Main> filterEven[1,2,3,4,5,6,7]
[1,3,5,7]
*Main> 
```

Conclusion:

- The filterEven function uses Haskell's built-in filter function to remove all even numbers from the input list, returning only the odd numbers.

4) Implement a function listZipWith that behaves similarly to zipWith in Haskell. It should take a function and two lists, and return a list by applying the function to corresponding elements from both lists. For example, given the function + and the lists [1, 2, 3] and [4, 5, 6], the result should be [5, 7, 9].

Objective:

To Implement a function listZipWith that behaves similarly to zipWith in Haskell. It should take a function and two lists, and return a list by applying the function to corresponding elements from both lists. For example, given the function + and the lists [1, 2, 3] and [4, 5, 6], the result should be [5, 7, 9].

Program Code:

```
asecomputerlab@asecomputerlab: ~/Documents/franav/lab4  ×  asecomputerlab@asecomputerlab: ~/Documents/franav/lab4
GNU nano 2.9.3                                                                                       qn4.hs

listZipWith :: (a -> b -> c) -> [a] -> [b] -> [c]
listZipWith _ [] [] = []
listZipWith f (x:xs) (y:ys) = f x y : listZipWith f xs ys
listZipWith _ _ _ = []
```

Explanation Of The Code:

- The type signature defines that the function `listZipWith` takes a function $(a \rightarrow b \rightarrow c)$ (which operates on two arguments of types `a` and `b` to produce a result of type `c`), followed by two lists: one of type `[a]` and one of type `[b]`, and it returns a list of type `[c]`.
- The function uses recursion:
- If both input lists are empty, it returns an empty list.
- If both lists have elements, it applies the function `f` to the head elements of the two lists (`x` and `y`), and then recursively processes the rest of the lists (`xs` and `ys`).
- If the lists have different lengths, the function will return an empty list (this case is covered by the final pattern matching).

Input/Output Examples:

Input: `(+)[1,2,3][2,3,4]` Output: `[3,5,7]`

Input: `(-)[1,2,3][2,3,4]` Output: `[-1,-1,-1]`

Screenshot:

```
*Main> :l qn4.hs
[1 of 1] Compiling Main                ( qn4.hs, interpreted )
Ok, modules loaded: Main.
*Main> listZipWith (+) [1,2,3][2,3,4]
[3,5,7]
*Main> listZipWith (-) [1,2,3][2,3,4]
[-1,-1,-1]
*Main> 
```

Conclusion:

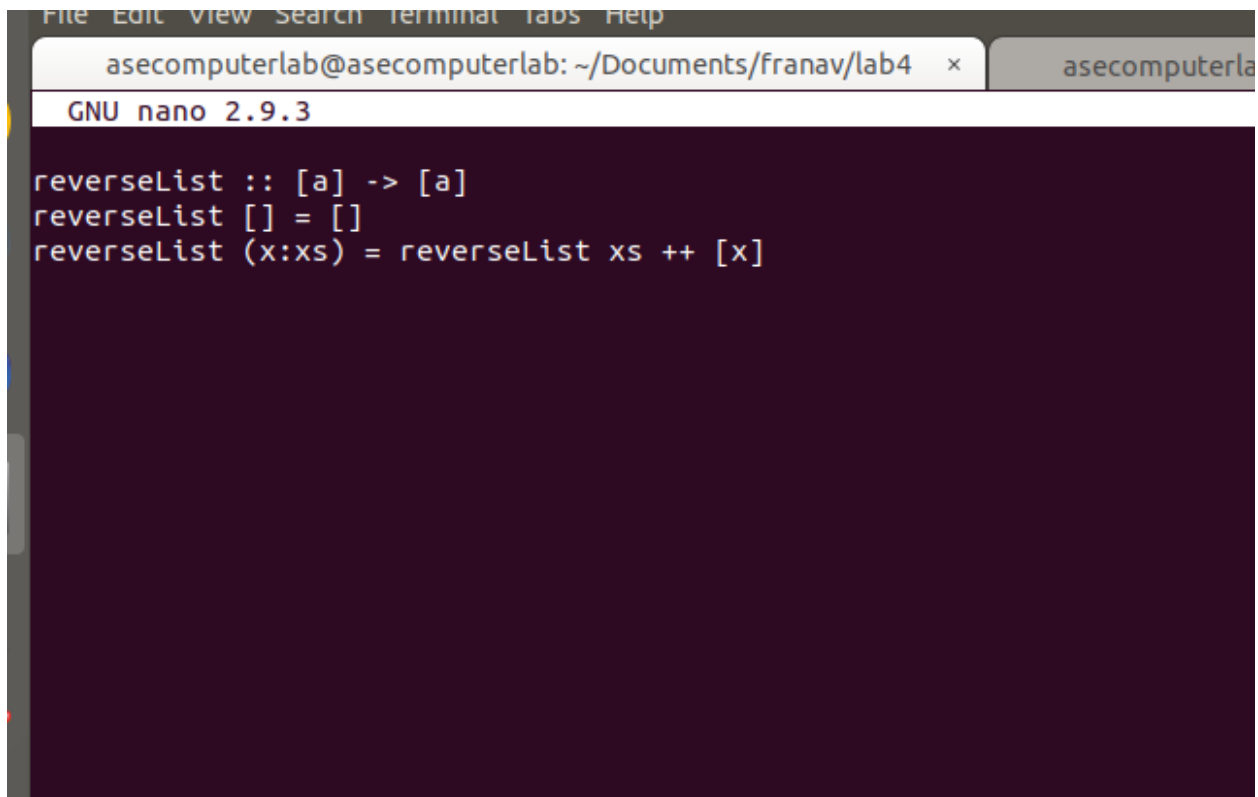
- The `listZipWith` function replicates the behavior of `zipWith`, applying a given function to corresponding elements from two input lists. It handles lists of equal and unequal lengths and returns a new list of the results.

5) Write a recursive function reverseList that takes a list of elements and returns the list in reverse order. For example, given [1, 2, 3], the output should be [3, 2, 1].

Objective:

To Write a recursive function `reverseList` that takes a list of elements and returns the list in reverse order. For example, given [1, 2, 3], the output should be [3, 2, 1].

Program Code:

A screenshot of a terminal window with a dark purple background. The window title bar shows 'asecomputerlab@asecomputerlab: ~/Documents/franav/lab4' and 'GNU nano 2.9.3'. The code displayed is:

```
reverseList :: [a] -> [a]
reverseList [] = []
reverseList (x:xs) = reverseList xs ++ [x]
```

Explanation of the Code:

- The type signature defines that the function `reverseList` takes a list of any type `a` and returns a list of the same type `a`.

- The function works recursively:
- If the input list is empty ([]), it returns an empty list (base case).
- If the input list has elements ((x:xs)), it reverses the rest of the list (reverseList xs) and then appends the head element x to the end of the reversed list (++ [x]).

Input/Output Examples:

Input: [1,2,3] Output:[3,2,1]

Input:[6,7,8,9] Output:[9,8,7,6]

Screenshot

```
*Main> :l qn5.hs
[1 of 1] Compiling Main                ( qn5.hs, interpreted )
Ok, modules loaded: Main.
*Main> reverseList [1, 2, 3]
[3,2,1]
*Main> reverseList [6,7,8,9]
[9,8,7,6]
*Main> █
```

Conclusion:

- The reverseList function recursively reverses the order of elements in a list by processing the head element and combining it with the reversed tail. It handles both non-empty and empty lists correctly.

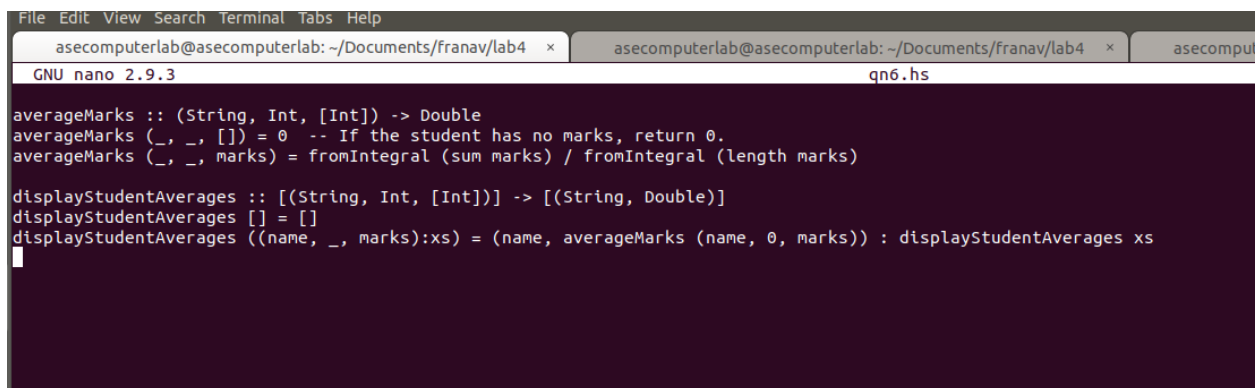
6)Write the Haskell Code to Write a recursive function averageMarks to calculate the average of a student's marks.The function should take a tuple containing a student's name, roll number, and marks list as

input.Display all student names along with their average marks.

Objective:

- Write a recursive function `averageMarks` to calculate the average of a student's marks.
- The function should take a tuple containing a student's name, roll number, and marks list as input.
- Display all student names along with their average marks.

Program Code:

A screenshot of a terminal window with a dark background. At the top, there are three tabs: 'asecomputerlab@asecomputerlab: ~/Documents/franav/lab4', 'asecomputerlab@asecomputerlab: ~/Documents/franav/lab4', and 'asecomputerlab@asecomputerlab: ~/Documents/franav/lab4'. Below the tabs, the text 'GNU nano 2.9.3' is visible on the left and 'qn6.hs' on the right. The main area of the terminal contains Haskell code:

```
averageMarks :: (String, Int, [Int]) -> Double
averageMarks (_, _, []) = 0 -- If the student has no marks, return 0.
averageMarks (_, _, marks) = fromIntegral (sum marks) / fromIntegral (length marks)

displayStudentAverages :: [(String, Int, [Int])] -> [(String, Double)]
displayStudentAverages [] = []
displayStudentAverages ((name, _, marks):xs) = (name, averageMarks (name, 0, marks)) : displayStudentAverages xs
```

Explanation of the Code:

- The type signature `averageMarks :: (String, Int, [Int]) -> Double` indicates that the function takes a tuple with a student's name, roll number, and marks, and returns a `Double` representing the average of the marks.
- Base Case: If the list of marks is empty (`[]`), the function returns `0`.
- Recursive Case: If there are marks, the function computes the sum of the marks and divides it by the number of subjects (length of the marks list), converting both the sum and length to `Double` for accurate division.
- `displayStudentAverages` takes a list of student tuples, computes the average for each student, and returns a list of tuples with student names and their average marks.

Input / Output Examples:

Input : let students = [("Pranav", 101, [90, 80, 85]), ("Kiruthik", 102, [70, 60, 75])]

Output: [("Pranav",85.0),("Kiruthik",68.33333333333333)]

Screenshot:

```
*Main> let students = [("Pranav", 101, [90, 80, 85]), ("Kiruthik", 102, [70, 60, 75])]  
*Main> print(displayStudentAverages students)  
[("Pranav",85.0),("Kiruthik",68.33333333333333)]  
*Main> █
```

Conclusion:

- The averageMarks function computes the average of a student's marks using a recursive approach. The displayStudentAverages function then processes a list of students, calculates their average marks, and displays the results.