# 20CYS312 -PRINCIPLE OF PROGRAMMING LANGUAGES

Date: 20-12-2024

Name: Naren S

Roll no.: CH.EN.U4CYS22034

GitHub: https://github.com/narenss/22034_20CYS312

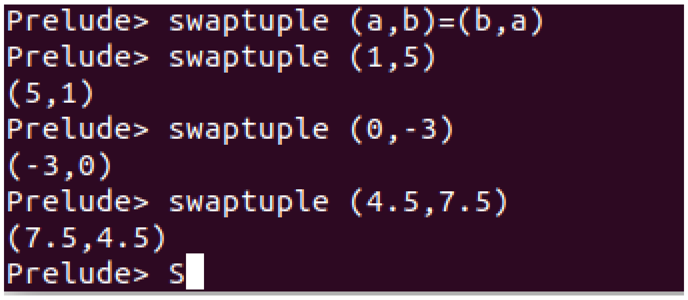
## LAB-4

## TUPLES AND LIST OPERATIONS:

Objective: Get familiar with basic data types like list and tuples.

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

Code and Output Examples:



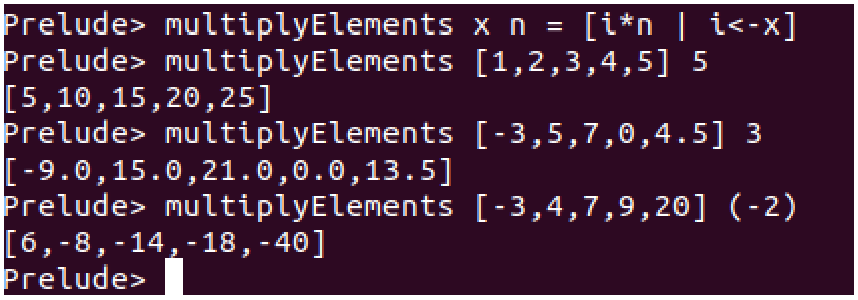
```
Prelude> swaptuple (a,b)=(b,a)
Prelude> swaptuple (1,5)
(5,1)
Prelude> swaptuple (0,-3)
(-3,0)
Prelude> swaptuple (4.5,7.5)
(7.5,4.5)
Prelude> S
```

**Explanation**:

- This function takes a tuple (x, y) as input, where x is of type a and y is of type b.
- The function returns a new tuple (y, x) where the elements are swapped. The types of the elements a and b are inferred by Haskell's type system.

Exercise 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.

Code and Output Examples:

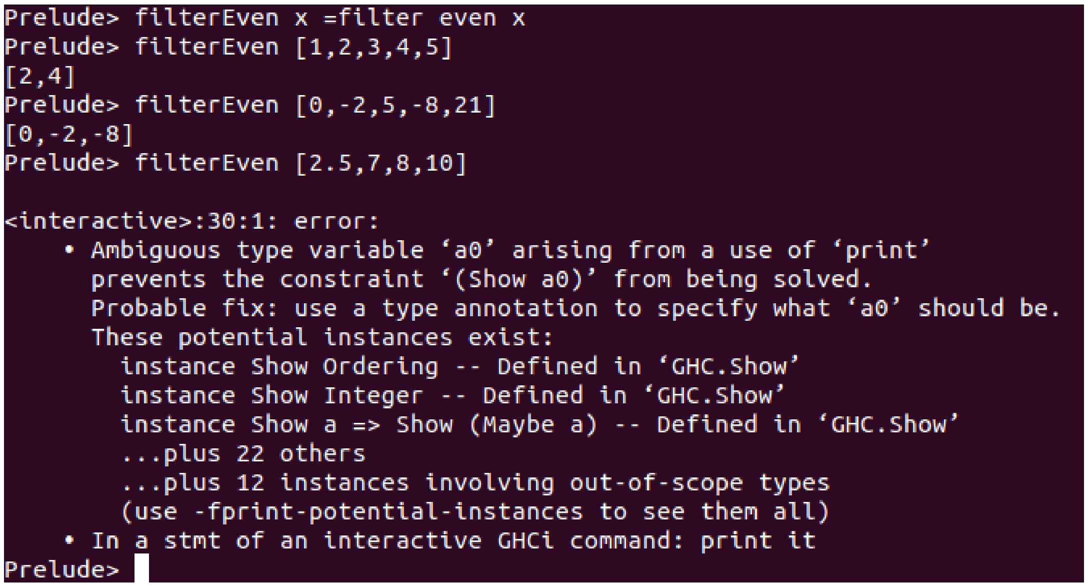

```
Prelude> multiplyElements x n = [i*n | i<-x]
Prelude> multiplyElements [1,2,3,4,5] 5
[5,10,15,20,25]
Prelude> multiplyElements [-3,5,7,0,4.5] 3
[-9.0,15.0,21.0,0.0,13.5]
Prelude> multiplyElements [-3,4,7,9,20] (-2)
[6,-8,-14,-18,-40]
Prelude>
```

**Explanation**:

- multiplyElements takes a list of integers xs and an integer n, then returns a new list where each element of xs is multiplied by n.
- **List Comprehension**: The list comprehension [x * n | x <- xs] iterates over each element x in the list xs and multiplies it by n. The result is a new list containing the products.

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

Code and Output Examples:

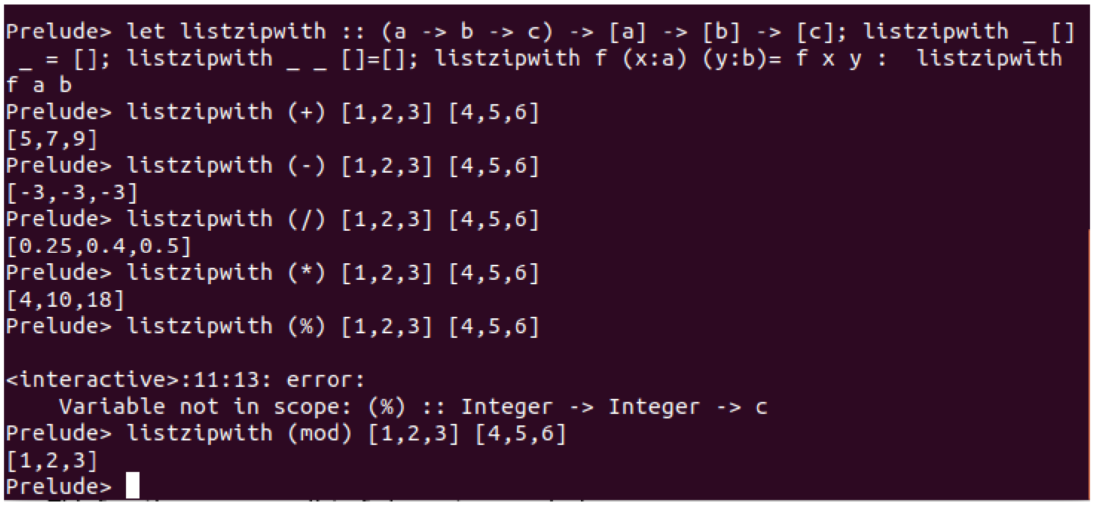

```
Prelude> filterEven x =filter even x
Prelude> filterEven [1,2,3,4,5]
[2,4]
Prelude> filterEven [0,-2,5,-8,21]
[0,-2,-8]
Prelude> filterEven [2.5,7,8,10]

<interactive>:30:1: error:
    • Ambiguous type variable ‘a0’ arising from a use of ‘print’
      prevents the constraint ‘(Show a0)’ from being solved.
      Probable fix: use a type annotation to specify what ‘a0’ should be.
      These potential instances exist:
        instance Show Ordering -- Defined in ‘GHC.Show’
        instance Show Integer -- Defined in ‘GHC.Show’
        instance Show a => Show (Maybe a) -- Defined in ‘GHC.Show’
        ...plus 22 others
        ...plus 12 instances involving out-of-scope types
        (use -fprint-potential-instances to see them all)
    • In a stmt of an interactive GHCi command: print it
Prelude>
```

Explaination:

- The filter function in Haskell takes a predicate and a list, and returns a list containing only the elements that satisfy the predicate.
- Here, the odd function is used as the predicate to filter out even numbers. The odd function returns True for odd numbers and False for even numbers, so only odd numbers remain in the list.

Exercise 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].

Code and Output Examples:



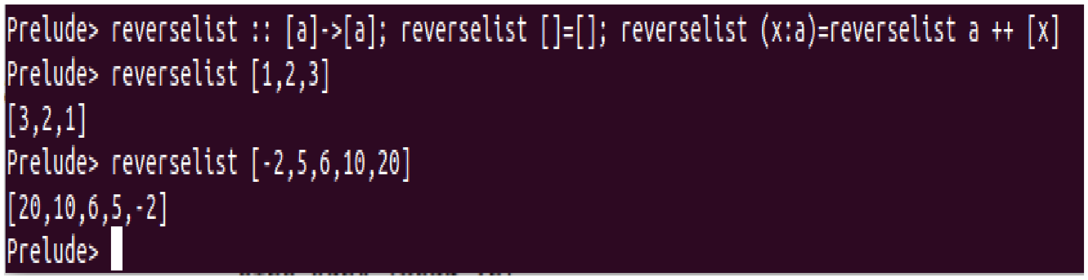
```
Prelude> let listzipwith :: (a -> b -> c) -> [a] -> [b] -> [c]; listzipwith _ []
 _ = []; listzipwith _ _ []=[]; listzipwith f (x:a) (y:b)= f x y :  listzipwith
f a b
Prelude> listzipwith (+) [1,2,3] [4,5,6]
[5,7,9]
Prelude> listzipwith (-) [1,2,3] [4,5,6]
[-3,-3,-3]
Prelude> listzipwith (/) [1,2,3] [4,5,6]
[0.25,0.4,0.5]
Prelude> listzipwith (*) [1,2,3] [4,5,6]
[4,10,18]
Prelude> listzipwith (%) [1,2,3] [4,5,6]

<interactive>:11:13: error:
    Variable not in scope: (%) :: Integer -> Integer -> c
Prelude> listzipwith (mod) [1,2,3] [4,5,6]
[1,2,3]
Prelude> 
```

Explaination:

- The function listZipWith is similar to zipWith in Haskell, which takes a binary function f, and two lists.
- **Base Case**: When both input lists are empty ([]), the result is an empty list.
- **Recursive Case**: When both lists have elements, the function applies the binary function f to the first elements of both lists (x and y), and then recursively processes the rest of the lists.
- The third clause, listZipWith _ _ _ = [], is a guard to handle cases where the lists are of different lengths. In such cases, the function returns an empty list.

Exercise 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].

Code and Output Examples:



```
Prelude> reverselist :: [a]->[a]; reverselist []=[]; reverselist (x:a)=reverselist a ++ [x]
Prelude> reverselist [1,2,3]
[3,2,1]
Prelude> reverselist [-2,5,6,10,20]
[20,10,6,5,-2]
Prelude> 
```

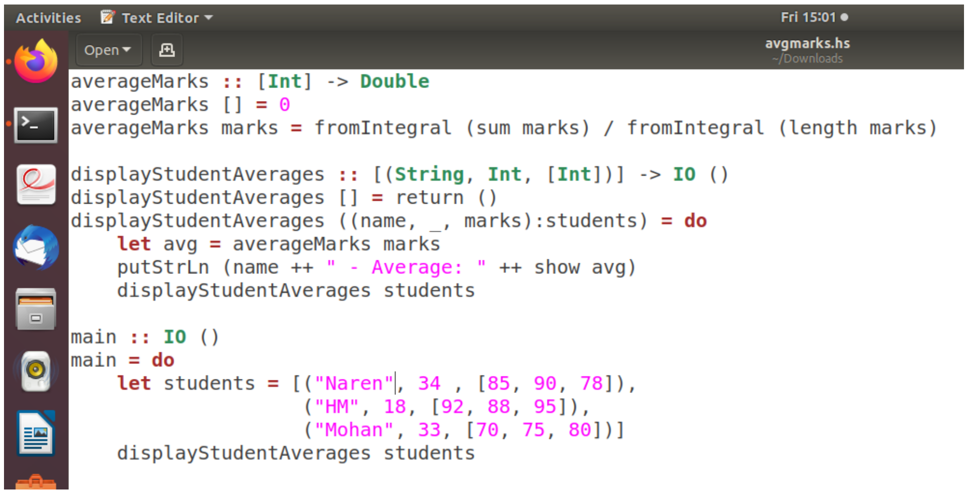
Explaination:

- This function reverses a list by recursively reversing the tail of the list (xs), and then appending the head element (x) to the reversed tail.
- **Base Case**: If the list is empty ([]), the reverse is also an empty list.
- **Recursive Case**: If the list is non-empty, the function reverses the tail (xs) and then appends the head (x) to the end.

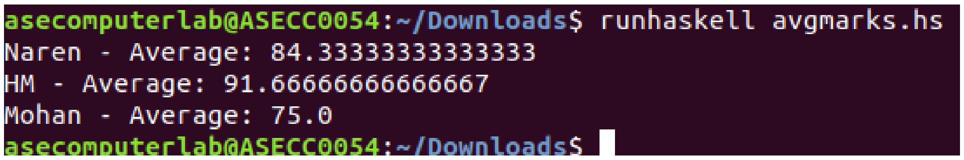
Exercise 6:

You are tasked with developing a program to manage and analyze student records. Each student is represented as a tuple (String, Int, [Int]), where the first element is the student's name (a string), the second is their roll number (an integer), and the third is a list of integers representing their marks in various subjects. Write a recursive function averageMarks to calculate the average of a student's marks. Display all student names and their average marks.

Code and Output Examples:



```haskell
averageMarks :: [Int] -> Double
averageMarks [] = 0
averageMarks marks = fromIntegral (sum marks) / fromIntegral (length marks)

displayStudentAverages :: [(String, Int, [Int])] -> IO ()
displayStudentAverages [] = return ()
displayStudentAverages ((name, _, marks):students) = do
    let avg = averageMarks marks
    putStrLn (name ++ " - Average: " ++ show avg)
    displayStudentAverages students

main :: IO ()
main = do
    let students = [("Naren", 34 , [85, 90, 78]),
                    ("HM", 18, [92, 88, 95]),
                    ("Mohan", 33, [70, 75, 80])]
    displayStudentAverages students
```



```
asecomputerlab@ASECC0054:~/Downloads$ runhaskell avgmarks.hs
Naren - Average: 84.33333333333333
HM - Average: 91.66666666666667
Mohan - Average: 75.0
asecomputerlab@ASECC0054:~/Downloads$
```

Explaination:

The function averageMarks computes the average of a student's marks.

- **Base Case**: If the list of marks is empty ([]), the function returns 0.
- **Recursive Case**: It calculates the sum of the marks (sum marks) and divides by the length of the list (length marks). The result is converted to Double using fromIntegral to avoid integer division.
- printStudentAverages uses the displayAverages function to generate the list of students' names and their average marks, then prints them.