AMRITA VISHWA VIDYAPEETHAM

NAME:SOWMYA K ROLL NUMBER:CH.EN.EN.U4CYS22046

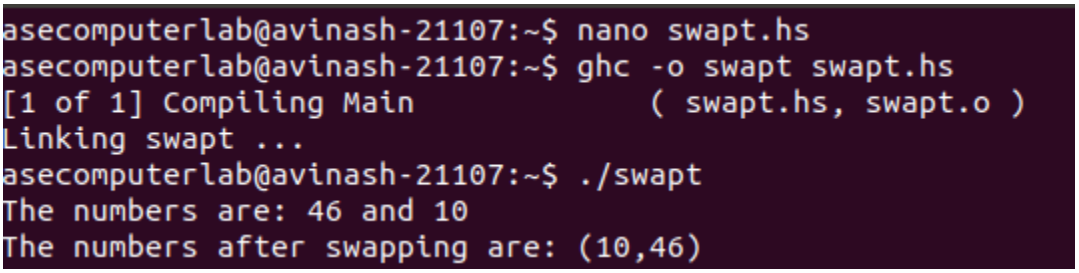
DATE:20.12.2024 COURSE CODE:20CYS312

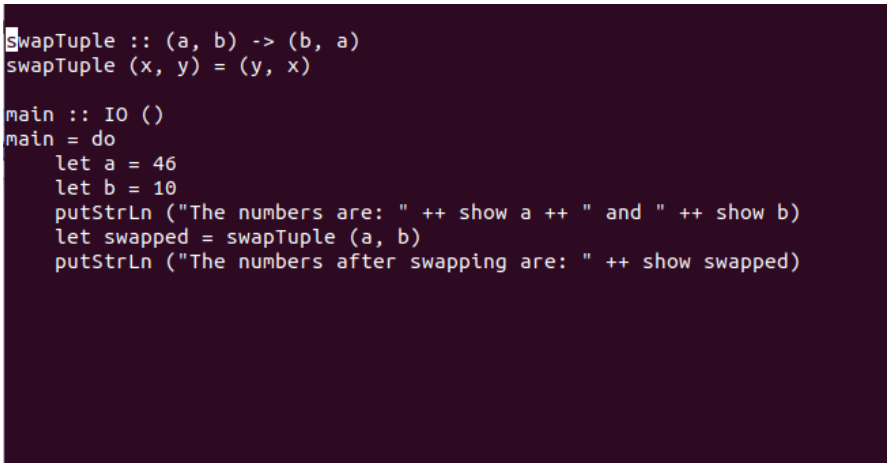
1.**OBJECTIVE:I**mplement a function swap Tuple that takes a tuple (a, b) and swaps its elements, i.e., returns the tuple (b, a) swapTuple :: (a, b) -> (b,A)

CODE:

```
main :: IO ()

main = do

  let a = 46

  let b = 10

  putStrLn ("The numbers are: " ++ show a ++ " and " ++ show b)

  let swapped = swapTuple (a, b)

  putStrLn ("The numbers after swapping are: " ++ show swapped)
```

SCREENSHOT:





EXPLANATION:

- swapTuple :: (a, b) -> (b, a): This function takes a tuple (x, y) where x is of type a and y is of type b, and returns a new tuple (y, x) where the elements are swapped. This is useful for swapping values in a tuple.

OUTPUT

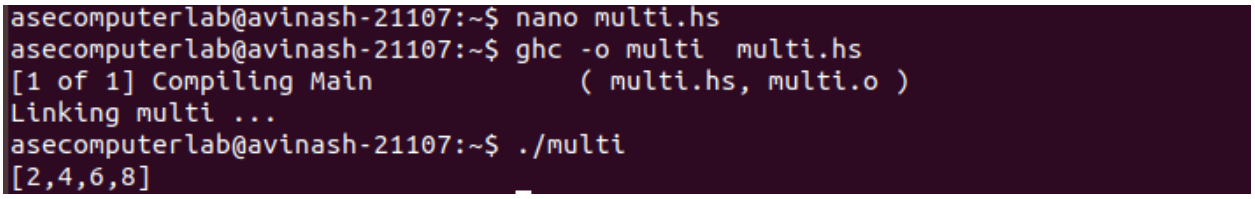
The numbers are: 46 and 10 The numbers after swapping are: (10,46)

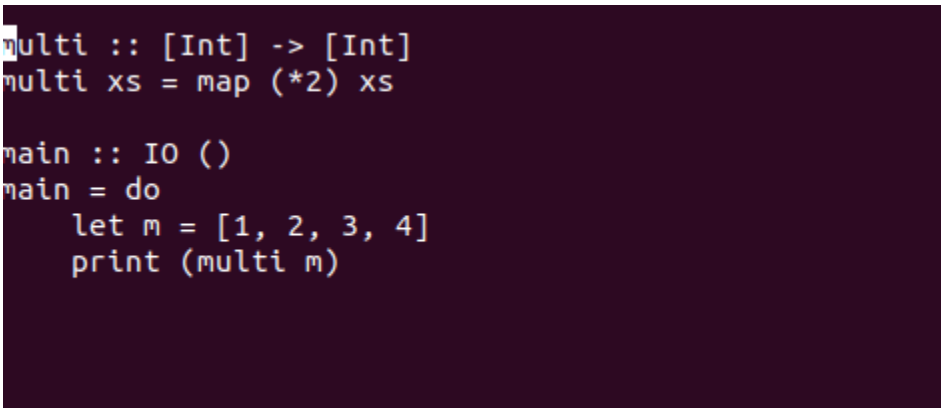
2. **OBJECTIVE:** Write a function multiplyElements that takes a list of numbers and a multiplier n, and returns a new list where each element is multiplied by n. Use a list comprehension for this task.

CODE:

```
multi :: [Int] -> [Int]

multi xs = map (*2) xs


main :: IO ()

main = do

  let m = [1, 2, 3, 4]

  print (multi m)
```

SCREENSHOT:





3.**OBJECTIVE:** Write a function filter Even that filters out all even numbers from a list of integers using the filter function

CODE:

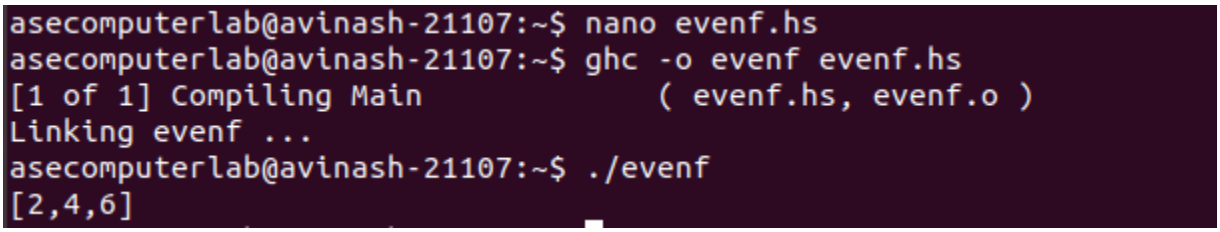
```
filterEven :: [Int] -> [Int]

filterEven xs = filter even xs


main :: IO ()

main = do

  let n = [1,2,3,4,5,6]

  print (filterEven n)
```

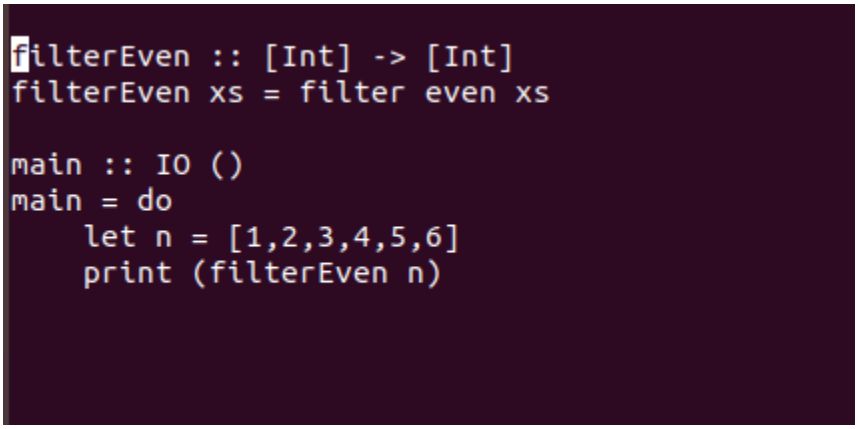
EXPLANATION:

- **map**: A higher-order function that applies a function to each element of a list. **(*2)**: A function that multiplies a number by 2, used in the map function to process each element of the list.

OUTPUT:
[2,4,6]

SCREENSHOT:





4**. OBJECTIVE:** Implement a function listZipWith that behaves similarly to zip With 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:

```
sumOfLists :: [Int] -> [Int] -> [Int]

sumOfLists xs ys = zipWith (+) xs ys


main :: IO ()

main = do

  let n = [1,2,3]

  let m = [1,2,3]

  let result = sumOfLists n m

  print result
```

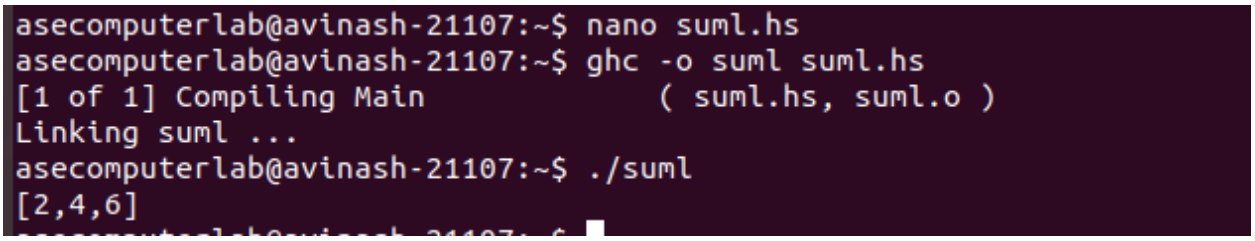
EXPLANATION:

zipWith (+) xs ys: The zipWith function takes a binary function (in this case, +) and two lists (xs and ys). It applies the function to corresponding elements from both lists and returns a new list with the results. Here, it adds the elements of xs and ys at the same position.

OUTPUT:

[2,4,6]

SCREENSHOT:



```
sumOfLists :: [Int] -> [Int] -> [Int]
sumOfLists xs ys = zipWith (+) xs ys

main :: IO ()
main = do
    let n = [1,2,3]
    let m = [1,2,3]
    let result = sumOfLists n m
    print result
```

5. **OBJECTIVE:** Write a recursive function reverse List 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:

```
main :: IO ()

main = do

  let tuple = [1, 2, 3]

  let result = reverse tuple

  print result
```

EXPLANATION:

let result = reverse tuple: The reverse function is used to reverse the list tuple. It will return a new list with the elements in reverse order.

OUTPUT:

[3, 2, 1]

```
asecomputerlab@avinash-21107:~$ nano reverse.hs
asecomputerlab@avinash-21107:~$ ghc -o reverse reverse.hs
[1 of 1] Compiling Main             ( reverse.hs, reverse.o )
Linking reverse ...
asecomputerlab@avinash-21107:~$ ./reverse
[3,2,1]
```

```
main :: IO ()
main = do
    let tuple = [1, 2, 3]
    let result = reverse tuple
    print result
```

6. **OBJECTIVE**: -- Define the student type as a tuple (Name, Roll Number, Marks List) type Student = (String, Int, [Int])

-- Recursive function to calculate the average marks of a student averageMarks :: [Int] -> Double averageMarks [] = 0 -- Base case: empty list, return 0 averageMarks marks = fromIntegral (sum marks) / fromIntegral (length marks)

-- Function to display student names and their average marks displayStudentAverages :: [Student] -> IO () displayStudentAverages [] = return () -- Base case: no students, do nothing displayStudentAverages ((name, _, marks):rest) = do let avg = averageMarks marks -- Calculate average for this student putStrLn $ name ++ ": " ++ show avg -- Display the student's name and average displayStudentAverages rest -- Recurse for the next student

-- Sample list of students students :: [Student] students = [("Alice", 101, [85, 90, 88]), ("Bob", 102, [78, 82, 79]), ("Charlie", 103, [92, 94, 89])]

CODE:

```
averageMarks :: [Int] -> Double

averageMarks [] = 0

averageMarks marks = fromIntegral (sum marks) / fromIntegral (length marks)



displayStudentAverages :: [(String, Int, [Int])] -> IO ()

displayStudentAverages [] = return ()

displayStudentAverages ((name, _, marks):rest) = do

  let avg = averageMarks marks

  putStrLn (name ++ ": " ++ show avg)

  displayStudentAverages rest

main :: IO ()

main = displayStudentAverages [("SOWMYA",14,[67,99,98]),

                ("DHARSHINI",46,[78,67,89]),

                ("HARSHITHA",51,[89,98,78])]
```

EXPLANATION:

The main function calls displayStudentAverages with a list of students. Each student is represented by a tuple of their name, roll number, and their list of marks.The program will display each student's name followed by their average marks.

OUTPUT:

SOWMYA: 88.0

DHARSHINI: 78.0

HARSHITHA: 88.33333333333333

SCREENSHOT:

```
asecomputerlab@avinash-21107:~$ nano student.hs
asecomputerlab@avinash-21107:~$ ghc -o student student.hs
[1 of 1] Compiling Main             ( student.hs, student.o )
Linking student ...
asecomputerlab@avinash-21107:~$ ./student
SOWMYA: 88.0
DHARSHINI: 78.0
HARSHITHA: 88.33333333333333
```

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

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

main :: IO ()
main = displayStudentAverages [("SOWMYA",14,[67,99,98]),
                               ("DHARSHINI",46,[78,67,89]),
                               ("HARSHITHA",51,[89,98,78])]
```