20CYS312 – PPL –LAB EXERCISE 3

Name: Kiruthik Pranav P V

Roll No:CH.EN.U4CYS22026

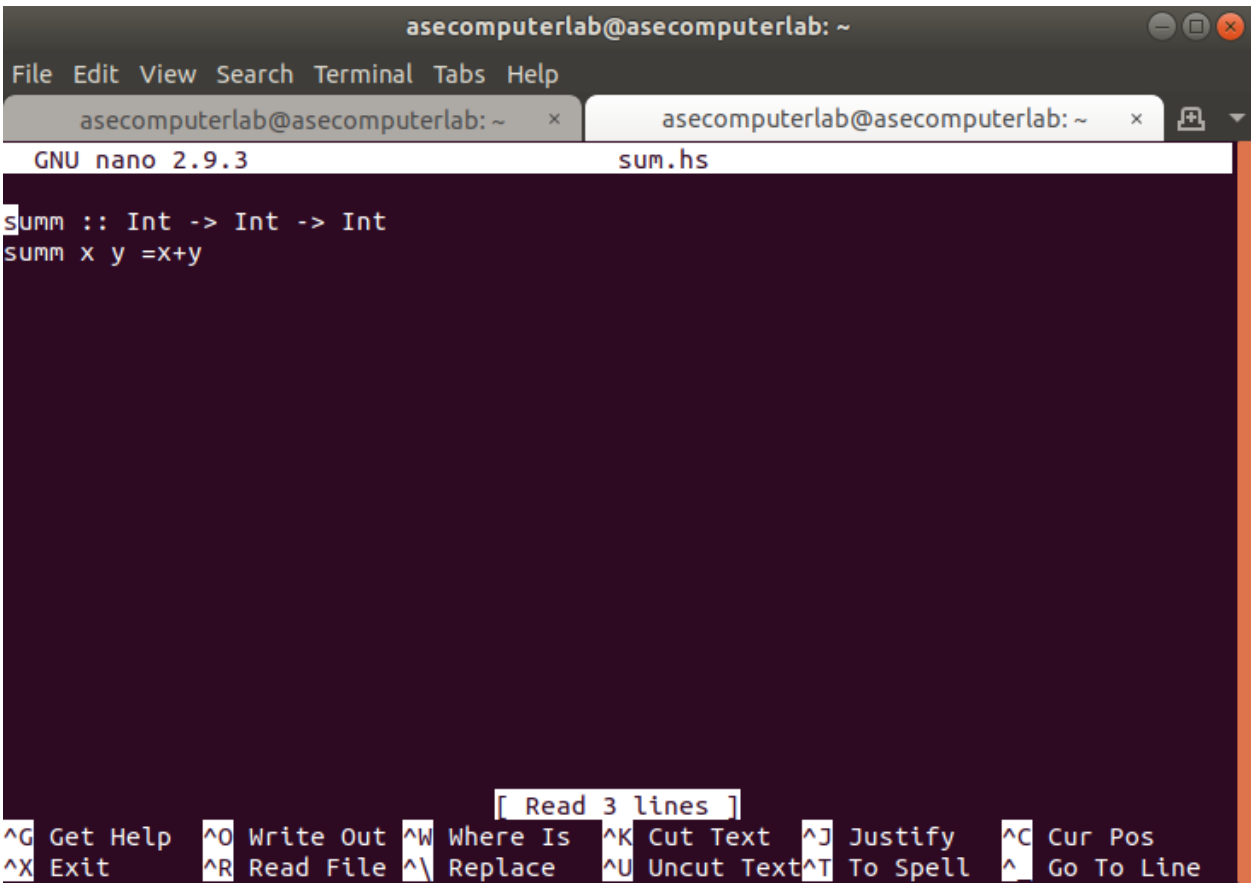
1.Basic Data Type

a)Write Haskell Function to perform the sum of two integers

Objective:

To define a function summ that takes two Int values and returns their sum.

Program Code:



Explanation Of the Code:

- We are telling that the first input is of int type and the second input is of int type and the resultant is of int type
- In the Second Line ,we are assigning variables for the input such as x y and we are adding them together and returning

Input/Output Examples:

Input:10 12 Output:22

Input:6 7 Output:12

Screenshot:

```
*Main> summ 10 12
22
*Main> summ 6 7
13
*Main>
```

Conclusion:

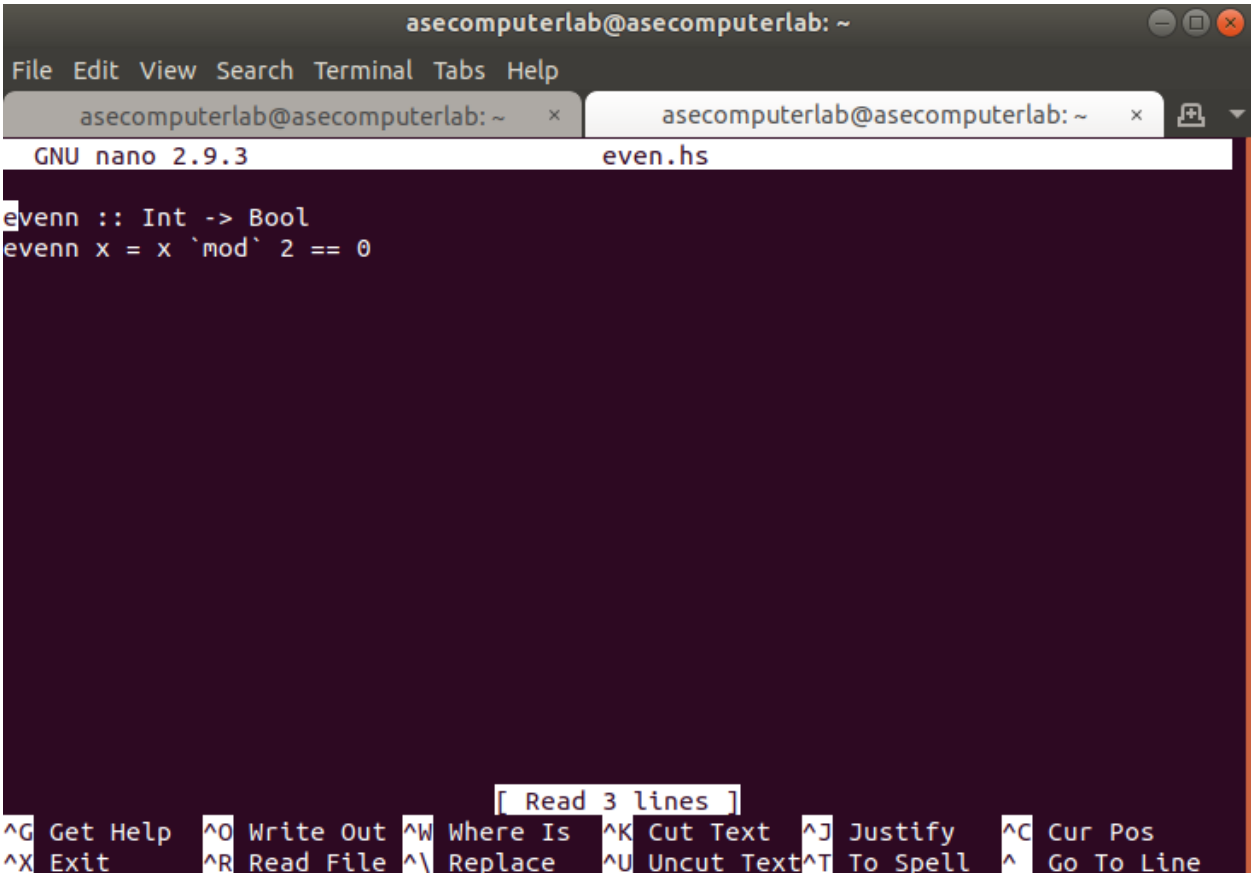
This function is simple, showing how we can define a function that adds two integers and returns their sum. It demonstrates how to handle input, perform arithmetic operations, and return results in Haskell.

b)Write Haskell Function to Check if a number is even or odd

Objective:

To write a function isEven that takes an Int and returns a Boolean value indicating whether the number is even.

Program Code:



Explanation Of the Code:

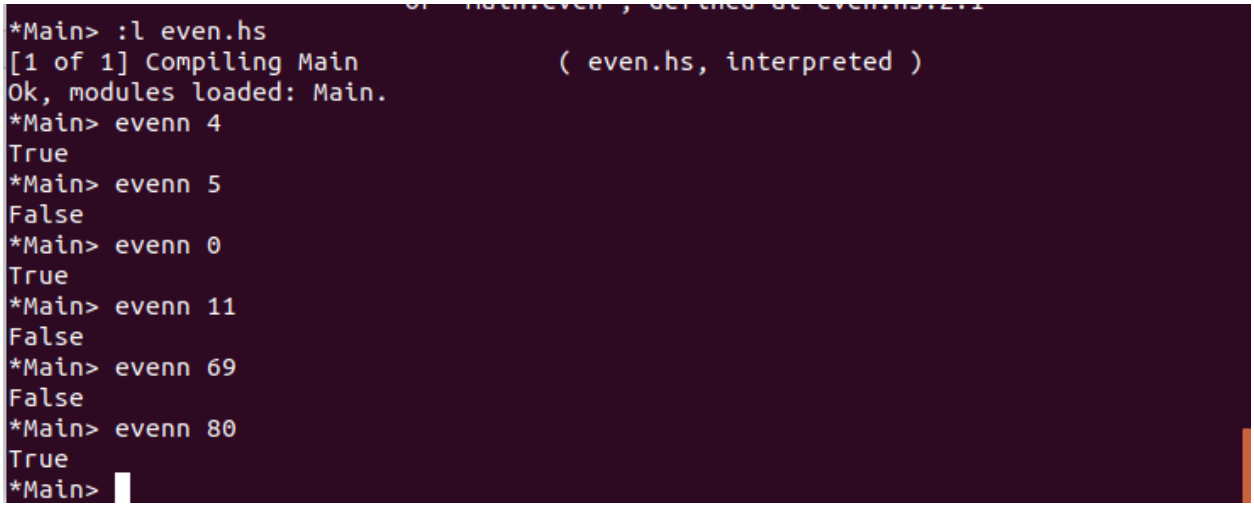
- We are telling that the input is of Int type and the output is of Bool type, where the function checks if the number is even.
- In the second line, we are assigning a variable n for the input. We then use the mod operator to divide n by 2 and check if the remainder is 0. If the remainder is 0, we return True, meaning the number is even. If the remainder is not 0, we return False, meaning the number is odd.

Input /Output Examples:

Input: 4 Output:True

Input:5 Output:False

Screenshot:



Conclusion:

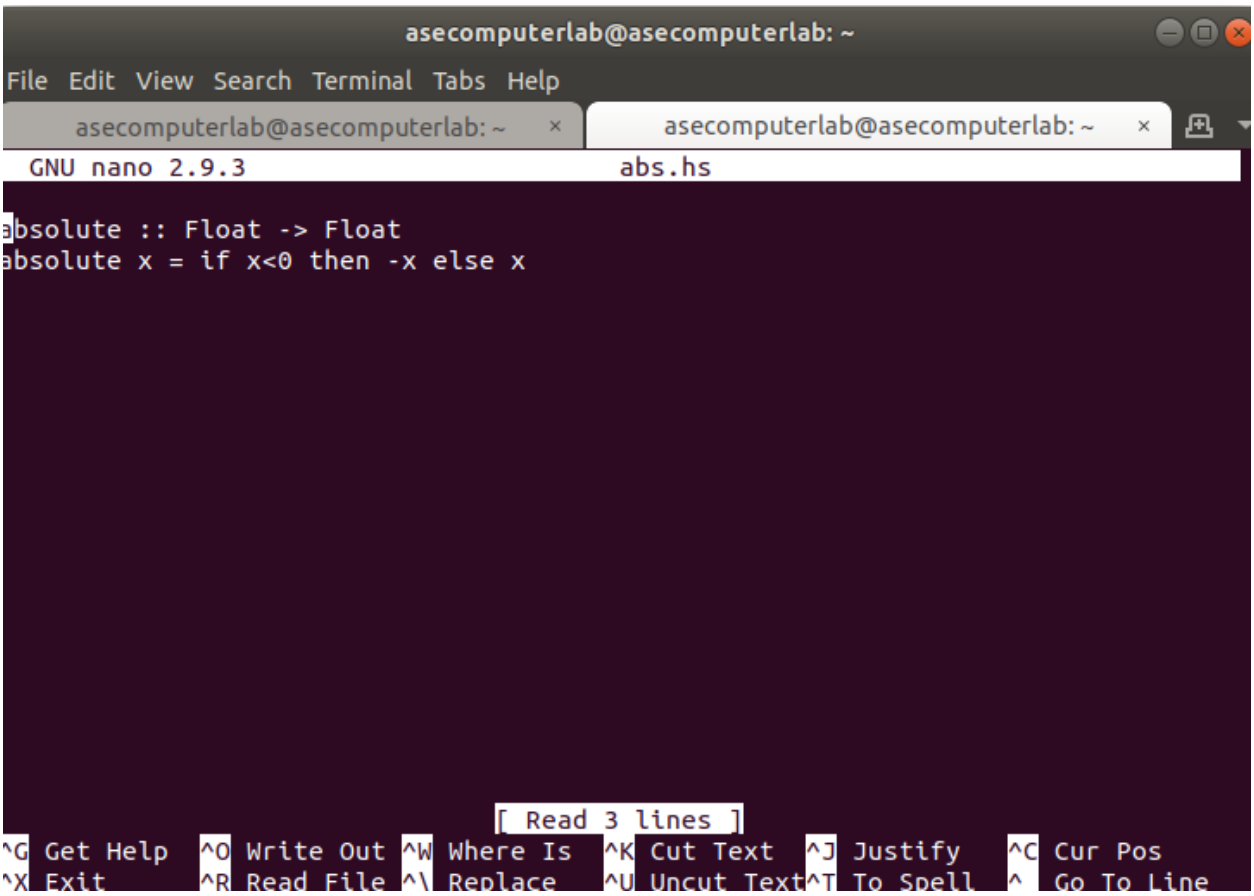
The function even takes an integer as input, checks if the remainder when divided by 2 is 0, and returns True if it is (indicating the number is even), or False if it's not (indicating the number is odd).

C)Write Haskell Function to return the absolute number of a given number

Objective:

To Define a function absolute that takes a Float and returns its absolute value.

Program Code:



Code Explanation:

- We are telling that the input is of Float type and the output is of Float type, where the function returns the absolute value of the number.
- In the second line, we are assigning a variable x for the input. We then check if x is less than 0. If it is, we negate x (turn it positive) by using -x. If x is already positive or zero, we just return x as it is.

Input / Output Examples:

Input:-12 Output:12.0

Input:12 Output:12.0

Screenshot:

```
*Main> absolute (-12)
12.0
*Main> absolute 12
12.0
*Main>
```

Conclusion:

- The absolute function takes a number, checks if it's negative, and converts it to a positive value if needed.

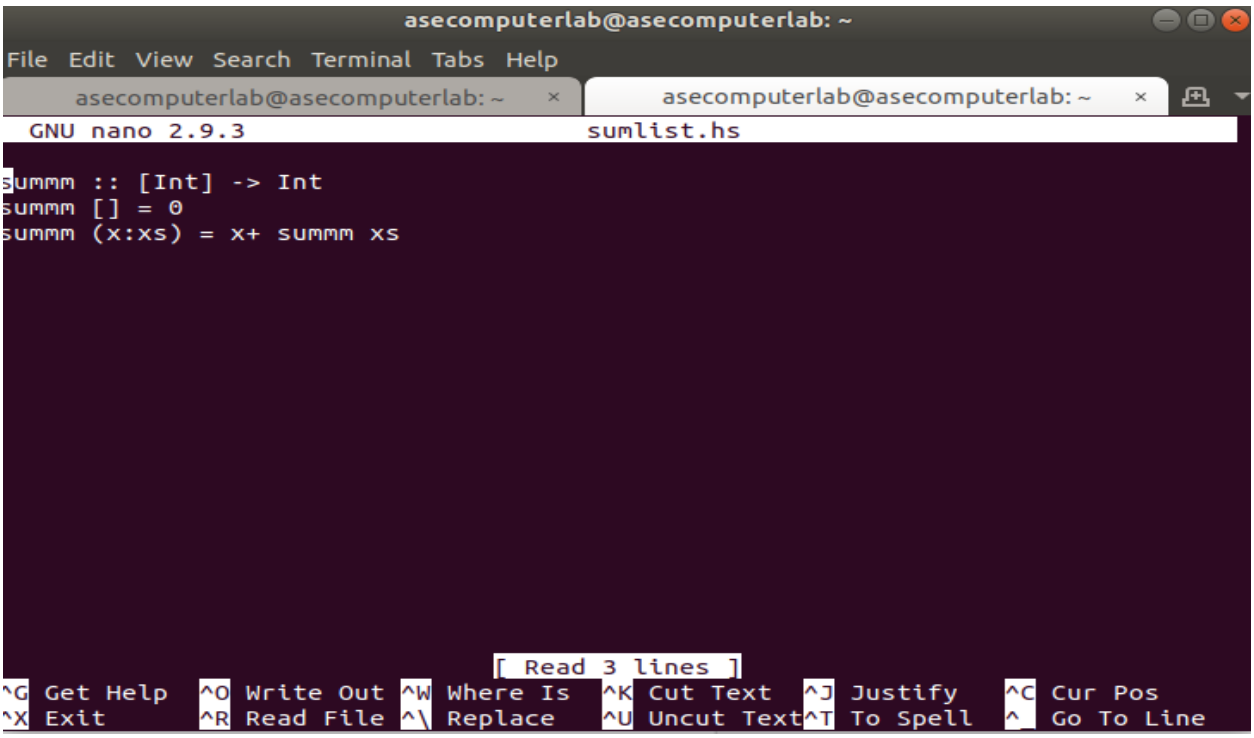
2.List Operations

A)Write the Haskell Code to find the sum of all elements

Objective:

To define a function sumList that takes a list of integers and returns the sum of all the elements in the list.

Program Code:



Explanation Of The Code:

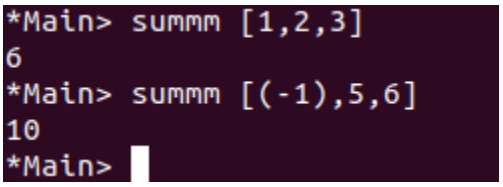
- We are telling that the input is a list of Int values and the output is an Int value, which is the sum of all the elements in the list.
- In the second line, we are checking if the list is empty. If it is empty, the sum is 0. If the list is not empty, we take the first element x and add it to the sum of the rest of the list by calling sumList recursively on the tail xs.

Input/Output Examples:

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

Input [-1,5,6] Output:10

Screenshot:



Conclusion:

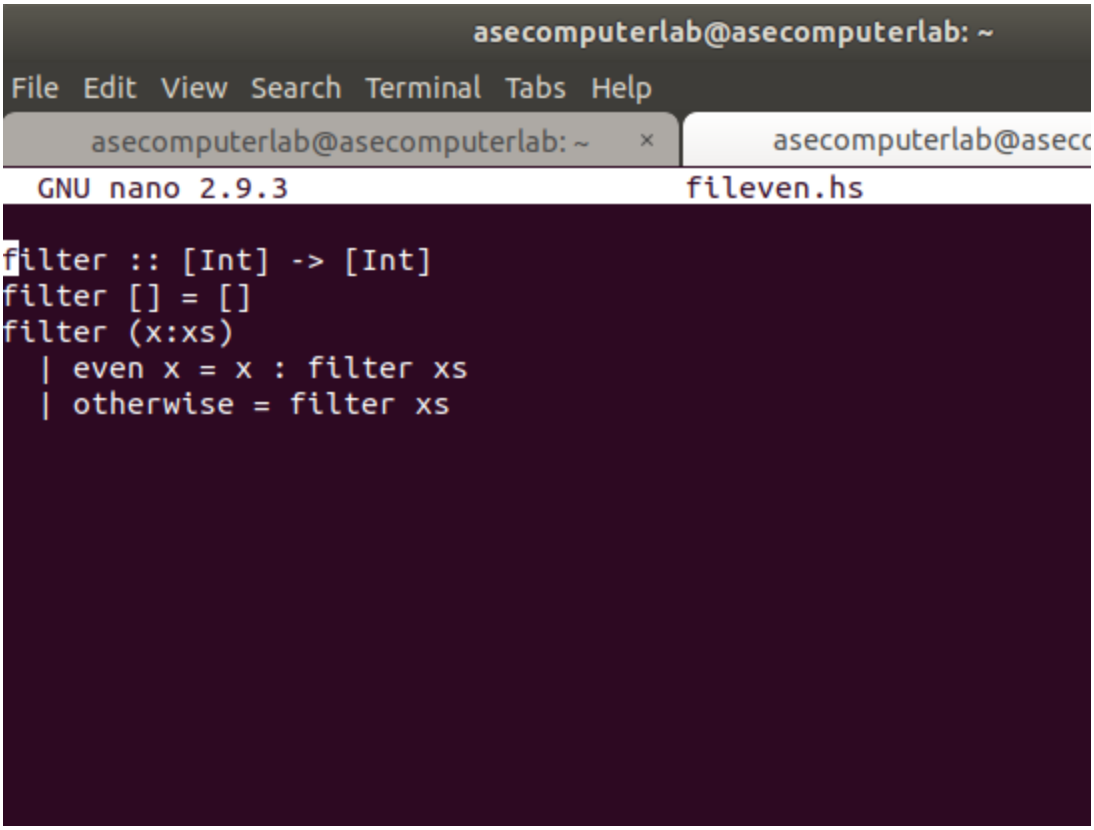
- The sumList function recursively adds up all the elements in a list by processing one element at a time. If the list is empty, the sum is 0, and if not, it adds the first element to the sum of the remaining elements. This approach makes use of recursion to break down the problem until it reaches the base case of an empty list.

B)Write the Haskell Code to filter all even numbers

Objective:

To write a function filterEven that takes a list of integers and returns a list containing only the even numbers.

Program Code:



Explanation of the Code:

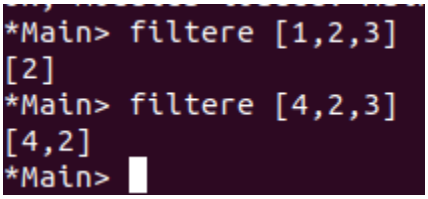
- We are telling that the input is a list of Int values and the output is a list of Int values, which contains only the even numbers from the original list.
- In the second line, we are checking if the list is empty. If it is empty, we return an empty list. If the list is not empty, we check the first element x:
- If x is even (using the even function), we include x in the result by prepending it to the filtered list of the rest of the elements.
- If x is not even (odd), we skip x and recursively process the rest of the list xs.

Input/Output Examples:

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

Input:[4,2,3] Output:[4,2]

Screenshot



Conclusion:

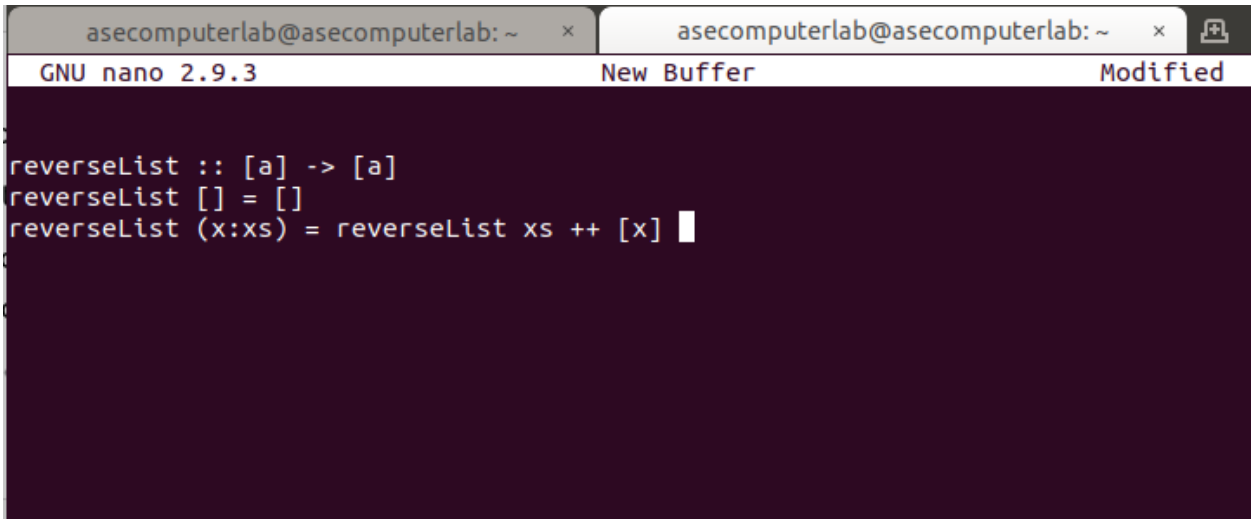
- The filtere function takes a list of integers and filters out the odd numbers, leaving only the even ones. It uses recursion to check each element, including only the even numbers in the resulting list. If the list is empty, it returns an empty list, and if the list contains even numbers, they are included in the final result.

C)Write the Haskell Code toreverse a list

Objective:

To define a function reverseList that takes a list and returns a new list with the elements in reverse order.

Program Code:



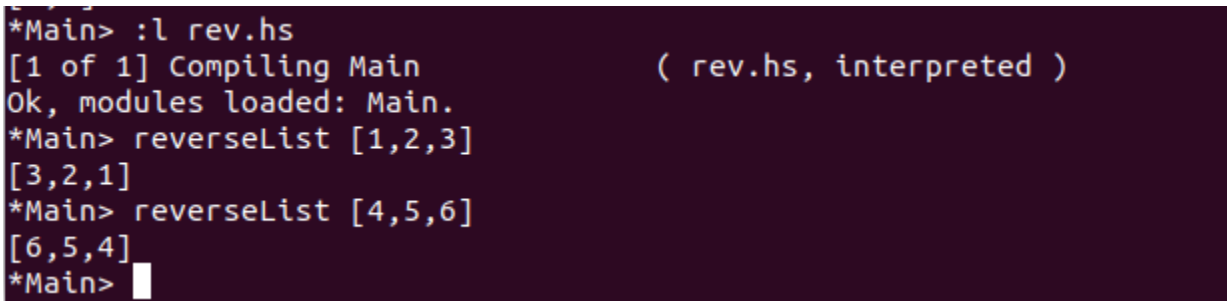
Explanation of the Code:

- The reverseList function reverses a list using recursion. It defines two cases: the base case is an empty list, which returns an empty list. In the recursive case, the function reverses the tail and then appends the head element to the reversed tail. This process continues until the entire list is reversed.

Input / Output Examples:

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

Screenshot:



Conclusion:

- The reverseList function takes a list and recursively processes it by reversing the rest of the list and appending the first element at the end. If the list is empty, it returns an empty list. If the list has elements, it reverses the tail of the list first and then adds the head at the end, ultimately reversing the entire list.

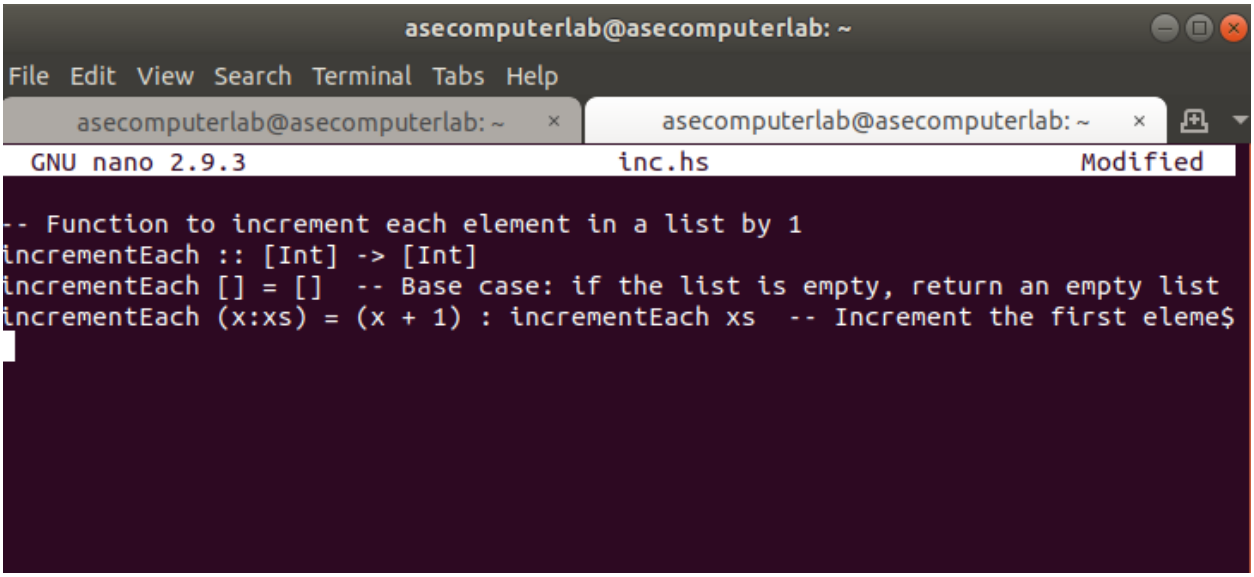
3)Basic Functions

A)Write the Haskell Code To Increment Each Element

Objective:

To define a function incrementEach that takes a list of integers and returns a new list where each element is incremented by 1.

Program Code:



Explanation of the Code:

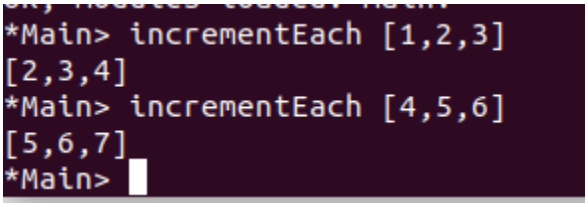
- We are telling that the input is a list of Int values, and the output is a new list of Int values, where each element is incremented by 1.
- In the second line, we are checking if the list is empty. If the list is empty ([]), we return an empty list because there are no elements to increment.
- If the list is not empty, we take the first element x, increment it by 1, and then recursively apply the incrementEach function to the rest of the list xs. We construct the new list by placing the incremented value (x + 1) at the front of the result.

Input/Output Examples:

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

Input:[4,5,6] Output:[5,6,7]

Screenshot:



Conclusion:

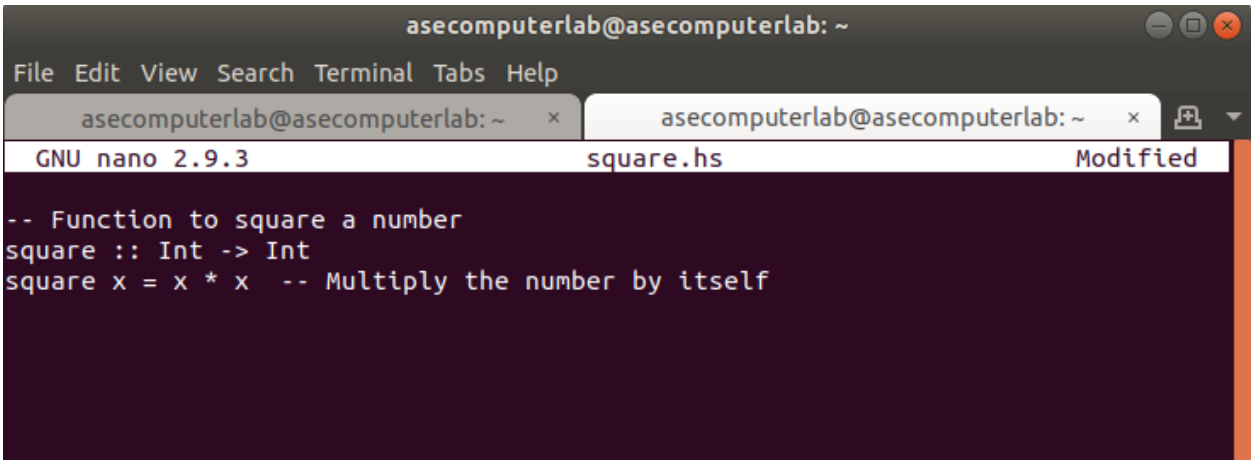
- The incrementEach function processes a list of integers by recursively incrementing each element by 1. If the list is empty, it returns an empty list. If the list is not empty, it increments the first element and continues recursively with the rest of the list. The final result is a new list where every element has been increased by 1. This is done efficiently using recursion and the basic list operations in Haskell.

B)Write the Haskell Code To Square a Number

Objective:

Write a function square that takes an integer and returns its square.

Program Code:



Explanation of the Code:

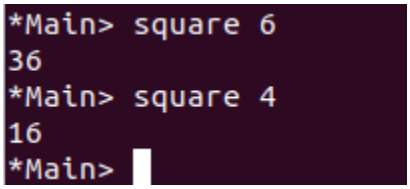
- The function square takes an integer x as input and returns its square, which is computed by multiplying x by itself (x * x).

Input/Output Examples:

Input: 6 Output:36

Input:4 Output:16

Screenshot:



Conclusion:

- The square function takes an integer as input and returns its square by multiplying the integer by itself. It is a straightforward operation that directly applies the formula x * x to calculate the square of the given number. This function effectively demonstrates how to perform basic arithmetic operations in Haskell.

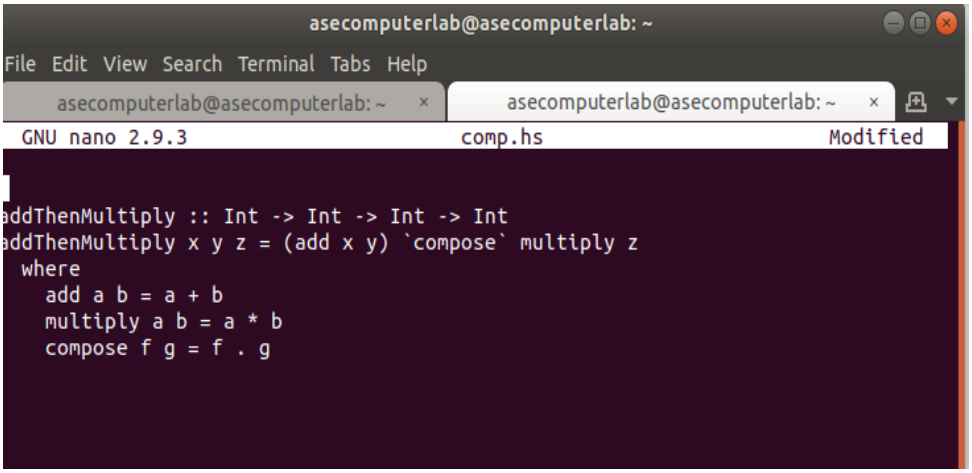
4)Function Composition

A)Write the Haskell Code To Compose functions to add and multiply

Objective:

Write a function addThenMultiply that first adds two integers and then multiplies the result by another integer. Use function composition to define this.

Program Code:



Explanation of the Code:

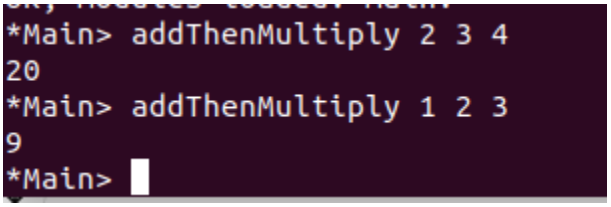
- We are telling that the input is three integers: x, y, and z. The output is the result of adding x and y, and then multiplying the sum by z.
- In the first line, we define a function addThenMultiply. We define two helper functions inside it:
- add a b = a + b to add two numbers together.
- multiply a b = a * b to multiply two numbers. Then we compose these two functions using the . operator, which allows us to chain the functions together: we first add x and y, and then multiply the result by z.

Input/Output Examples:

Input: 2 3 5 Output:20

Input:1 2 3 Output: 9

Screenshot:



Conclusion:

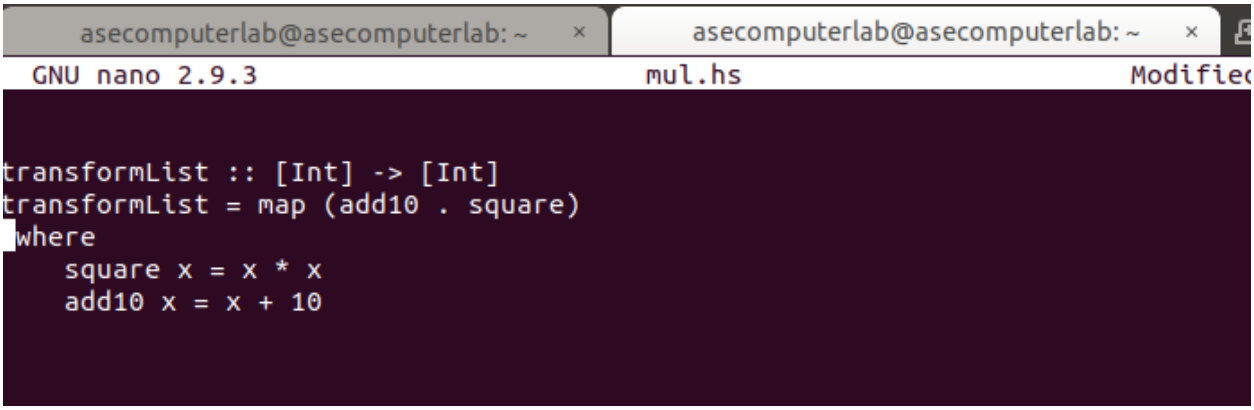
- The addThenMultiply function uses function composition to combine the addition and multiplication steps into one operation. First, it adds two integers, and then the result is multiplied by another integer. This demonstrates how you can chain functions to perform multiple operations in sequence using composition in Haskell.

B)Write the Haskell Code To apply multiple transformations

Objective:

To define a function transformList that takes a list of integers and first squares each element and then adds 10 to each squared element. Use function composition to implement this.

Program Code:



Explanation of the Code:

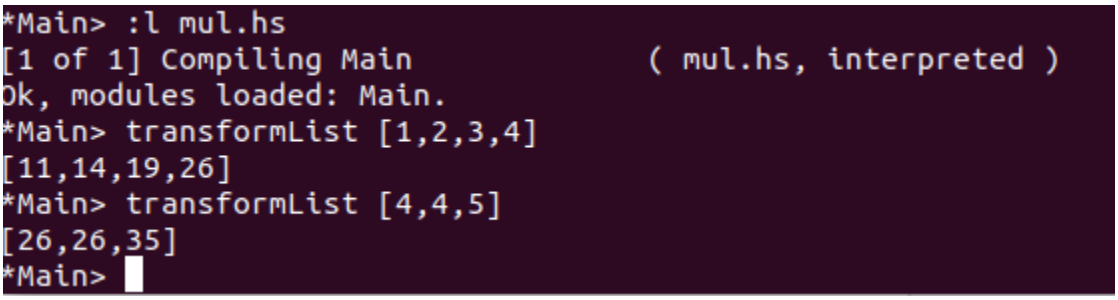
- We are telling that the input is a list of integers, and the output is a new list where each element is first squared and then increased by 10.
- In the second line, we define the transformList function. We use function composition with the map function.
- First, the square function is applied to each element in the list, which squares the element.
- After that, the add10 function is applied, which adds 10 to the squared result. Using map (add10 . square), we apply both transformations to each element in the list.

Input/Output Examples:

Input: transformList [1, 2, 3, 4] Output: [11, 14, 19, 26]

Input: transformList[4,4,5] Output:[26,26,35]

Screenshot:



Conclusion:

- The transformList function uses function composition to apply multiple transformations to each element in a list. It first squares each element, then adds 10 to the result.