20CYS312 - Principles of Programming Languages

S. Shyam Balaji **CH.EN.U4CYS22045** **DATE:** 13/12/24

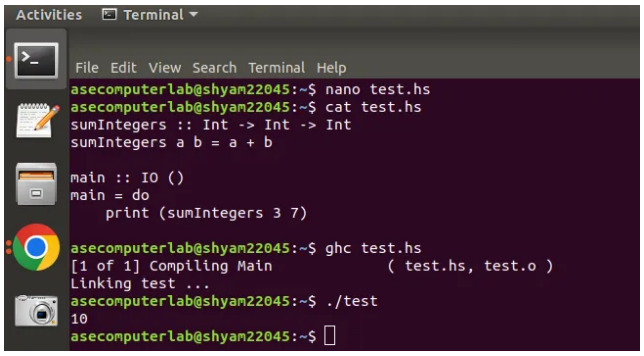
1. Basic Data Types

a. Sum of Two Integers

Objective:

To define a function that calculates the sum of two integers and returns the result.

Function Definition:



Explanation:

The function `sumIntegers` takes two arguments of type `Int` and returns their sum.

Example:

```
sumIntegers 4 6
-- Output: 10
```

Conclusion:

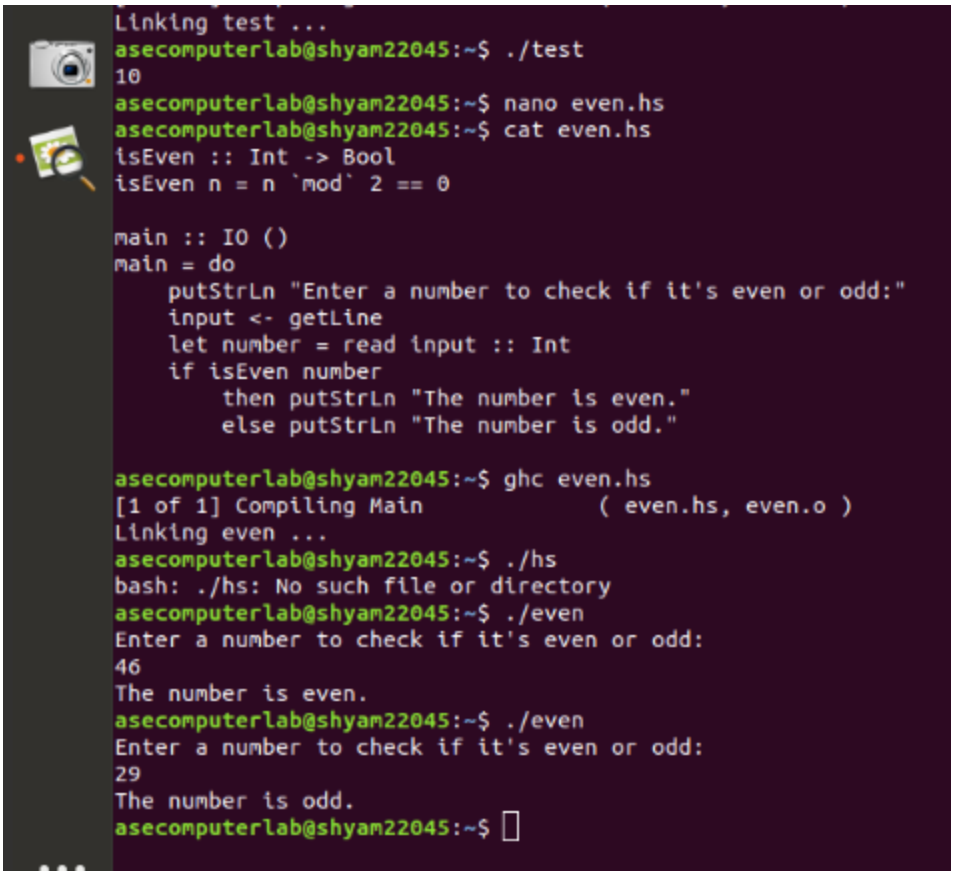
This function demonstrates a simple arithmetic operation in Haskell.

b. Check if a Number is Even or Odd

Objective:

To define a function that checks if a given number is even.

Function Definition:



Explanation:

The `isEven` function determines whether a number is divisible by 2 without a remainder.

Example:

```
isEven 46
-- Output: True
```

Conclusion:

The function showcases how to perform logical checks using Haskell.

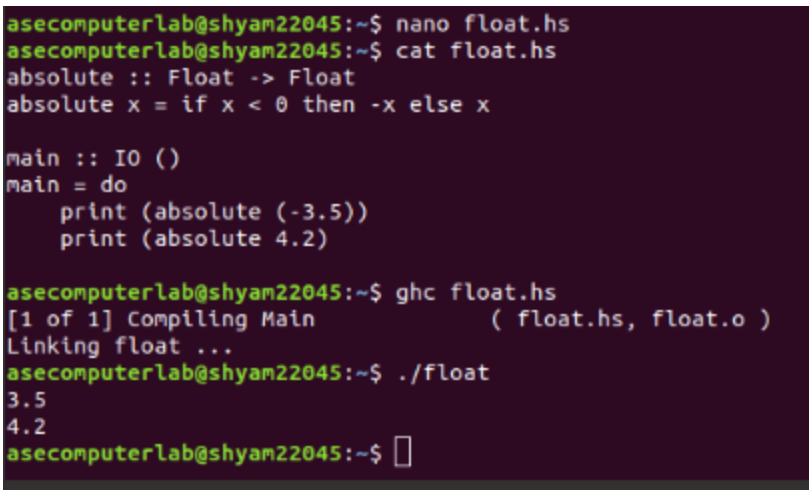
---

c. Absolute Value

Objective:

To define a function that computes the absolute value of a given number.

Function Definition:



Explanation:

The `absolute` function checks if a number is negative and converts it to its positive equivalent.

Example:

```
absolute (-3.5)
-- Output: 3.5
```

Conclusion:

This function demonstrates conditional logic in Haskell.

---

2. List Operations

a. Sum of All Elements

Objective:

To compute the sum of all elements in a list.

Function Definition:

```
asecomputerlab@shyam22045:~$ nano sumlist.hs
asecomputerlab@shyam22045:~$ cat sumlist.hs
sumList :: [Int] -> Int
sumList xs = sum xs

main :: IO ()
main = do
    print (sumList [1, 2, 3, 4, 5])
    print (sumList [10, -3, 7])

asecomputerlab@shyam22045:~$ ghc sumlist.hs
[1 of 1] Compiling Main             ( sumlist.hs, sumlist.o )
Linking sumlist ...
asecomputerlab@shyam22045:~$ ./sumlist
15
14
asecomputerlab@shyam22045:~$
```

Explanation:

The `sumList` function utilizes Haskell's built-in `sum` function to add all integers in a list.

Example:

```
sumList [1, 2, 3, 4, 5]
sumList [10, -3, 7]
-- Output: 15
-- Output: 14
```

Conclusion:

This function efficiently calculates the sum of elements in a list.

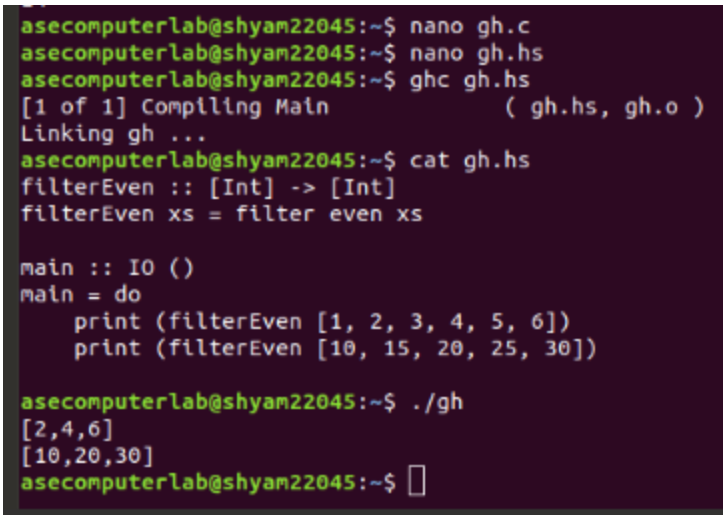
---

b. Filter Even Numbers

Objective:

To filter out only even numbers from a given list.

Function Definition:



Explanation:

The function `filterEven` applies the `filter` function with the predicate `even` to keep only even numbers.

Example:

```
filterEven [1, 2, 3, 4, 5, 6]
-- Output: [2, 4, 6]
```

Conclusion:

This function demonstrates list filtering in Haskell.

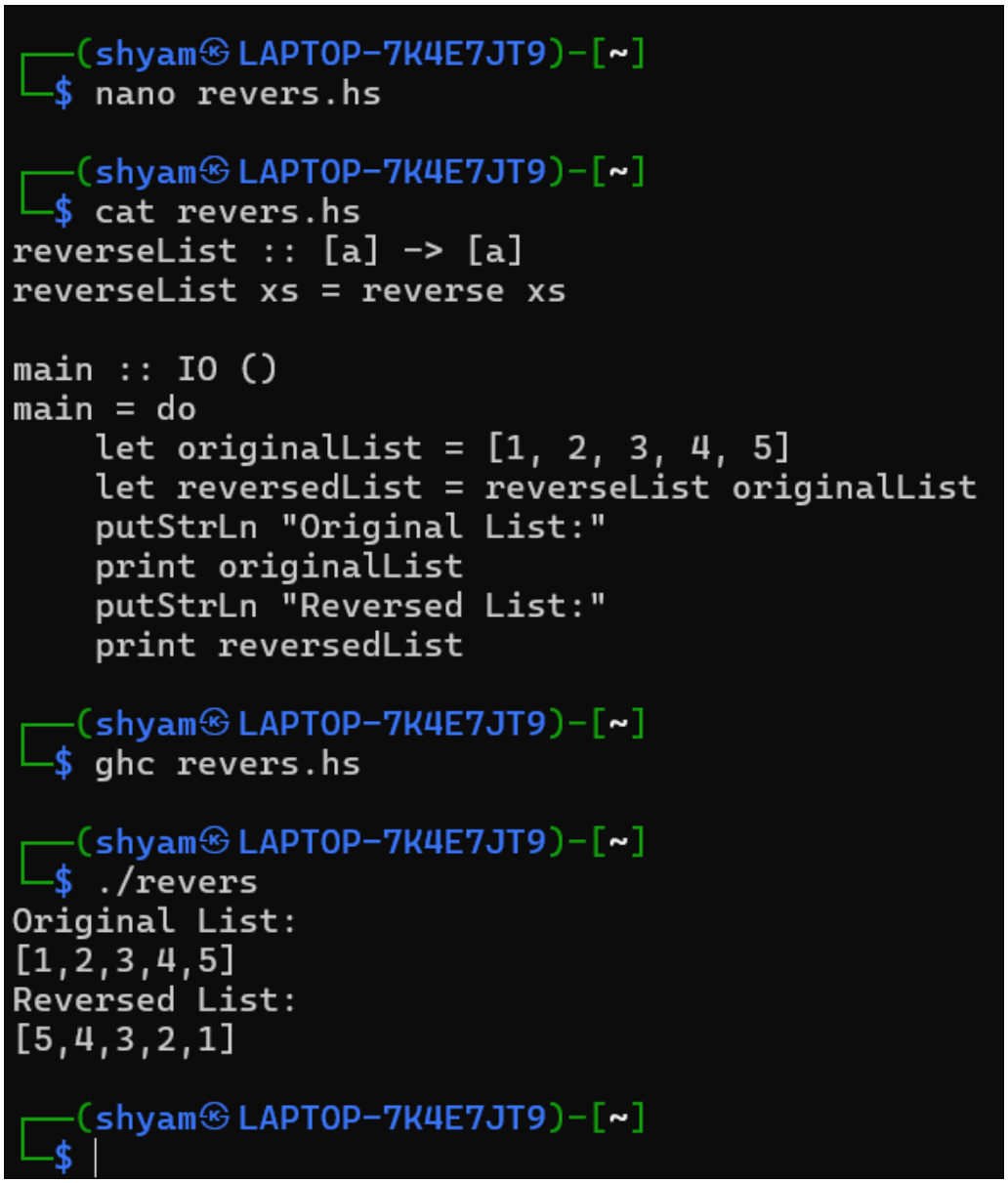
---

c. Reverse a List

Objective:

To reverse the order of elements in a list.

Function Definition:



Explanation:

The function `reverseList` uses the built-in `reverse` function to reorder the elements of a list.

Example:

```
reverseList [1, 2, 3, 4, 5]
-- Output: [5, 4, 3, 2, 1]
```

Conclusion:

This function showcases how to manipulate the order of list elements.

---

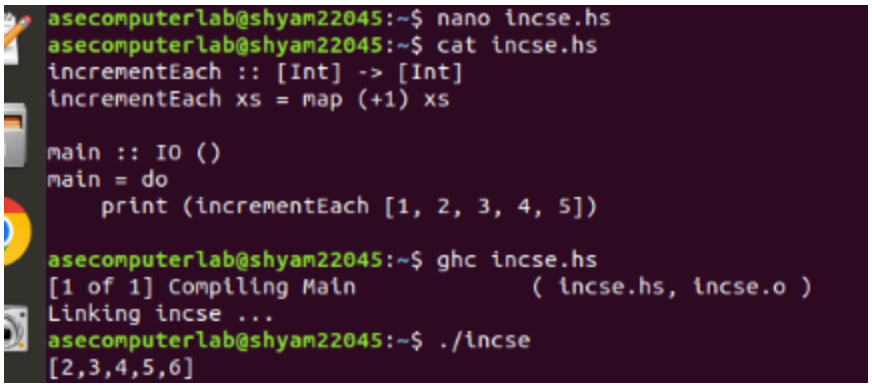
3. Basic Functions

a. Increment Each Element

Objective:

To increment each element in a list by 1.

Function Definition:



Explanation:

The `incrementEach` function uses `map` to apply the addition of 1 to each element of the list.

Example:

```
incrementEach [1, 2, 3, 5. 6]
-- Output: [2, 3, 4, 5, 6]
```

Conclusion:

This function illustrates how to use `map` for element-wise operations.

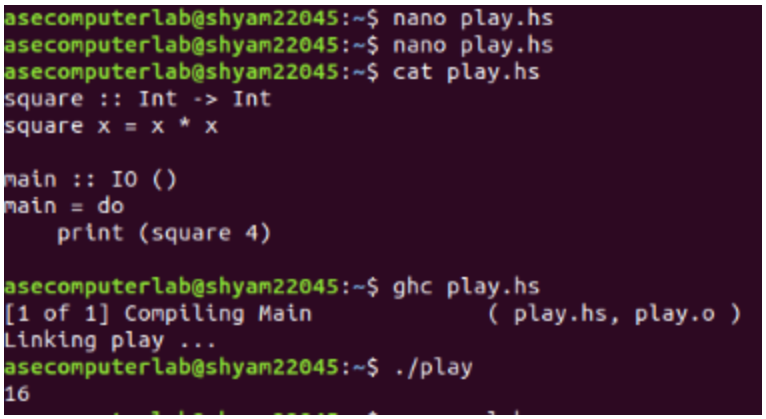
---

b. Square a Number

Objective:

To calculate the square of a given number.

Function Definition:



Explanation:

The `square` function multiplies a number by itself to find its square.

Example:

```
square 4
-- Output: 16
```

Conclusion:

This function highlights basic arithmetic operations in Haskell.

---

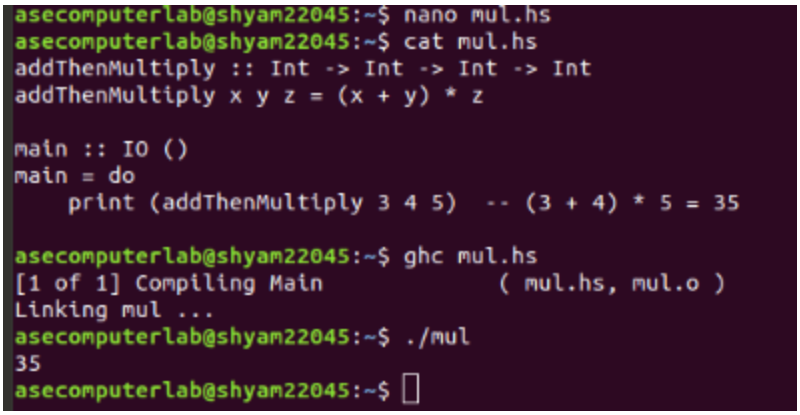
4. Function Composition

a. Compose Functions to Add and Multiply

Objective:

To compose two functions that add and multiply numbers.

Function Definition:



Explanation:

The function first adds `x` and `y` and then multiplies the result by `z`.

Example:

```
addThenMultiply 3 4 5
-- Output: 35
```

Conclusion:

This function demonstrates the power of combining operations in Haskell.

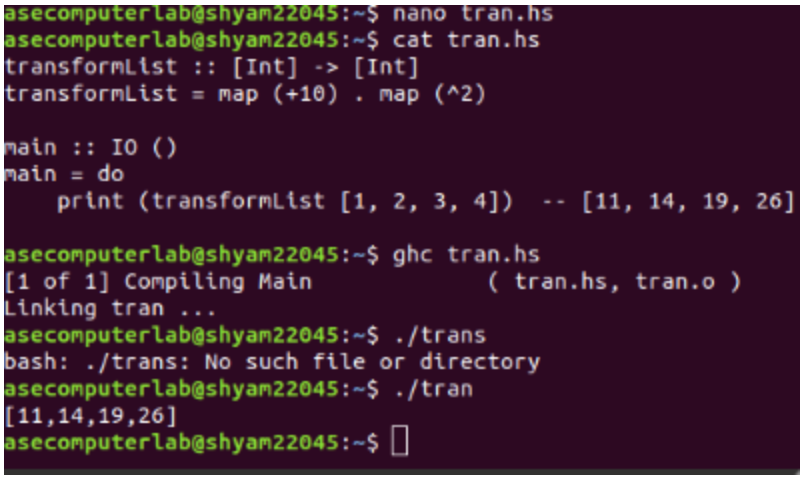
---

b. Apply Multiple Transformations

Objective:

To apply multiple transformations to elements in a list.

Function Definition:



Explanation:

The `transformList` function first squares each element and then adds 10 using a lambda function.

Example:

```
transformList [1, 2, 3] -- [11, 14, 19, 26]
-- Output: [11, 14, 19, 26]
```

Conclusion:

This function combines mathematical transformations using `map`.