20CYS312 -PRINCIPLE OF PROGRAMMING LANGUAGES

Date: 10-01-2025

Name: Naren S

Roll no.: CH.EN.U4CYS22034

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

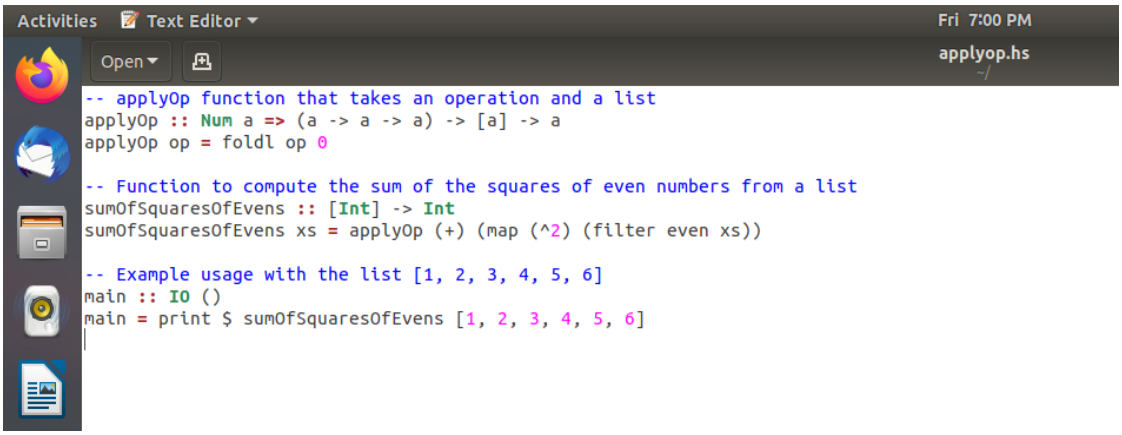
LAB-6

1. Currying, Map, and Fold

Objective: Get familiar with functions like Currying, map and fold.

Problem: Write a curried function applyOp that takes an operation (addition or multiplication) and a list of numbers, then applies the operation to the list and returns the final result. Use this function to compute the sum of the squares of all even numbers from the list [1, 2, 3, 4, 5, 6]. You must first filter out the even numbers, square them, and then compute the sum.

Code:



Output:

```
asecomputerlab@linux:~$ gedit applyop.hs
asecomputerlab@linux:~$ ghc -o applyop applyop.hs
[1 of 1] Compiling Main             ( applyop.hs, applyop.o )
Linking applyop ...
asecomputerlab@linux:~$ ./applyop
56
```

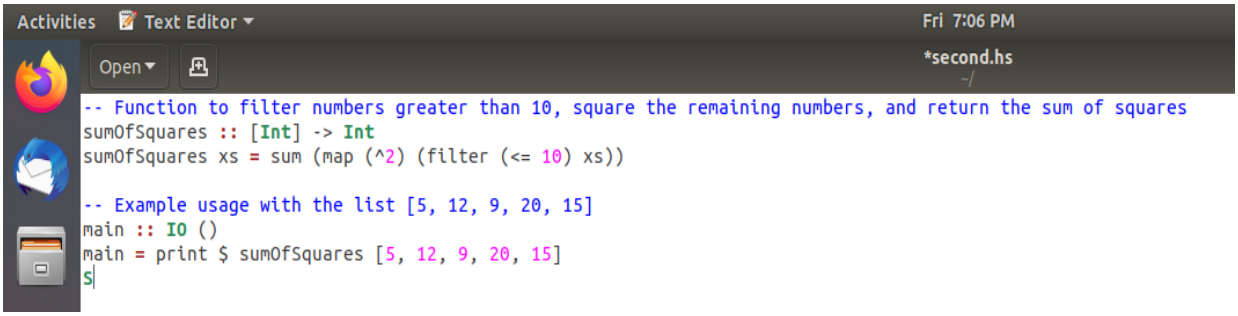
Explaination:

- foldl is a left fold that starts with an initial value (0 in this case) and applies the operation (op) to each element of the list.
- **Constraint (Num a):** It ensures the elements of the list are numeric (e.g., Int, Float).

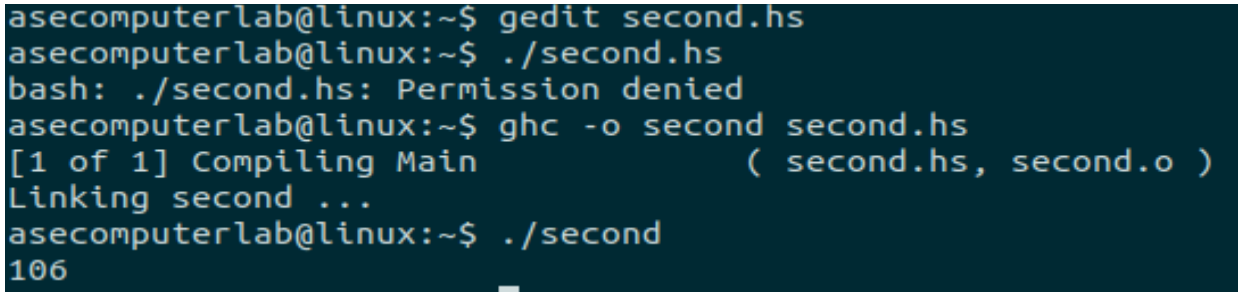
2. Map, Filter, and Lambda

Problem: Write a function that filters out all numbers greater than 10 from the list [5, 12, 9, 20, 15], then squares the remaining numbers and returns the sum of these squares. Use map and filter together, and apply the required transformations.

Code:



Output:



Explaination:

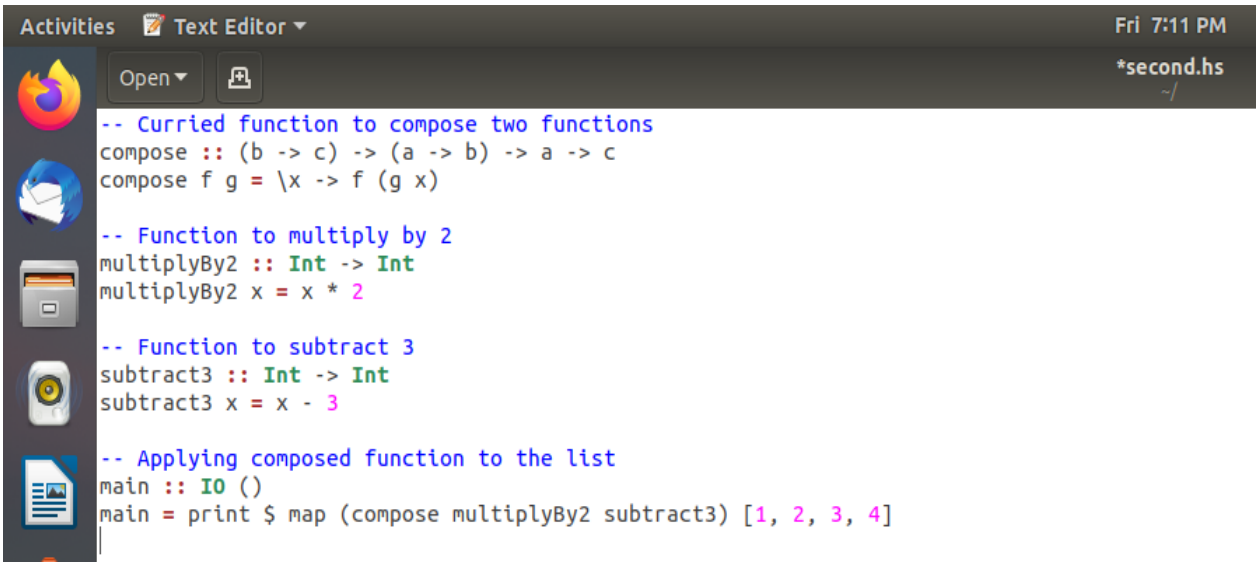
- **filter (<= 10) xs**:
  This filters out numbers that are greater than 10.
  It keeps only numbers that are less than or equal to 10.
  In this case, it filters the list [5, 12, 9, 20, 15] to [5, 9].
- **map (^2)**:
  This applies the squaring operation (^2) to each element of the filtered list.
  For [5, 9], it squares each element, resulting in [25, 81].
- **sum**:
  The sum function calculates the sum of the squared numbers.

It computes 25 + 81 = 106.

3. Currying, Function Composition, and Map

Problem: Write a curried function compose that takes two functions and returns their composition. Use this function to compose the following operations: multiply a number by 2, and then subtract 3 from the result. Apply this composed function to each element in the list [1, 2, 3, 4].

Code and Output Examples:




```
asecomputerlab@linux:~$ gedit second.hs
asecomputerlab@linux:~$ ghc -o second second.hs
[1 of 1] Compiling Main             ( second.hs, second.o )
Linking second ...
asecomputerlab@linux:~$ ./second
[-4,-2,0,2]
```

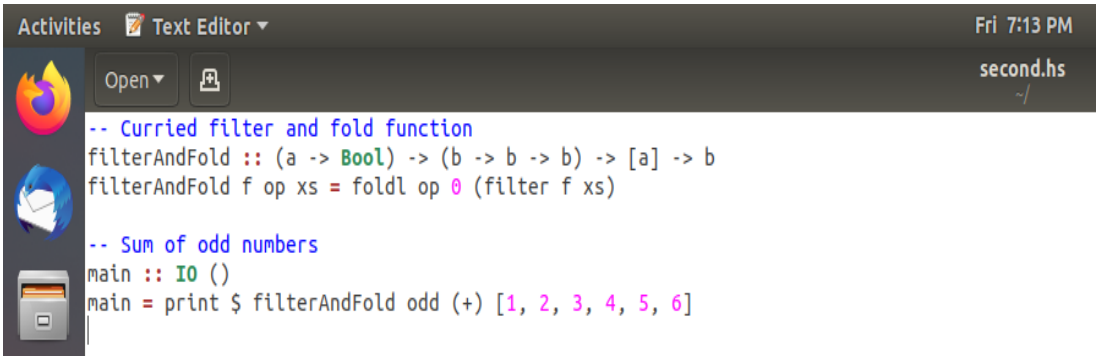
Explanation:

- compose combines two functions (multiplyBy2 and subtract3).
- We apply the composed function to each element of the list [1, 2, 3, 4] using map.
- Output: [-1, 1, 3, 5].

4. Currying, Filter, and Fold:

Problem: Write a curried function filterAndFold that takes a filtering function, a folding function, and a list. The function should first filter the list using the filtering function, and then apply the folding function to compute a result. Use this function to compute the sum of all odd numbers in the list [1, 2, 3, 4, 5, 6].

Code and Output Examples:




```
Naren@NAREN MINGW64 ~
$ nano fourth.hs

Naren@NAREN MINGW64 ~
$ ghc -o fourth fourth.hs
[1 of 2] Compiling Main             ( fourth.hs, fourth.o )
[2 of 2] Linking fourth.exe

Naren@NAREN MINGW64 ~
$ ./fourth
The sum of all odd numbers in the list is: 9
```

Explanation:

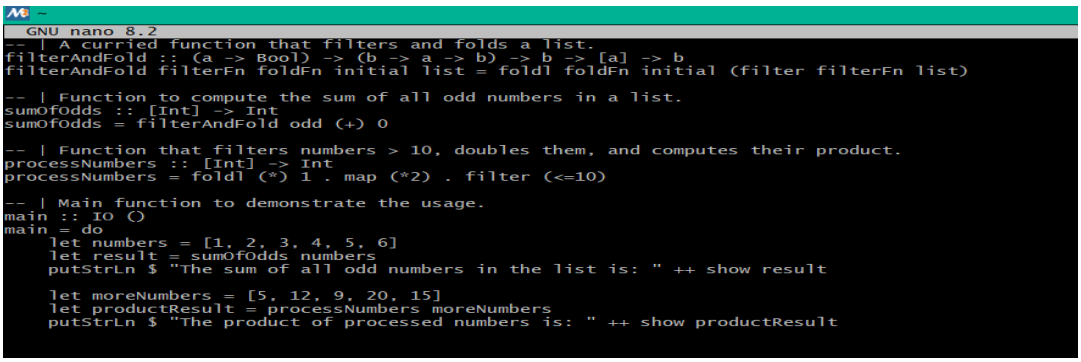
The program defines a function, `filterAndFold`, that filters a list based on a condition and then combines the filtered elements using a folding function.

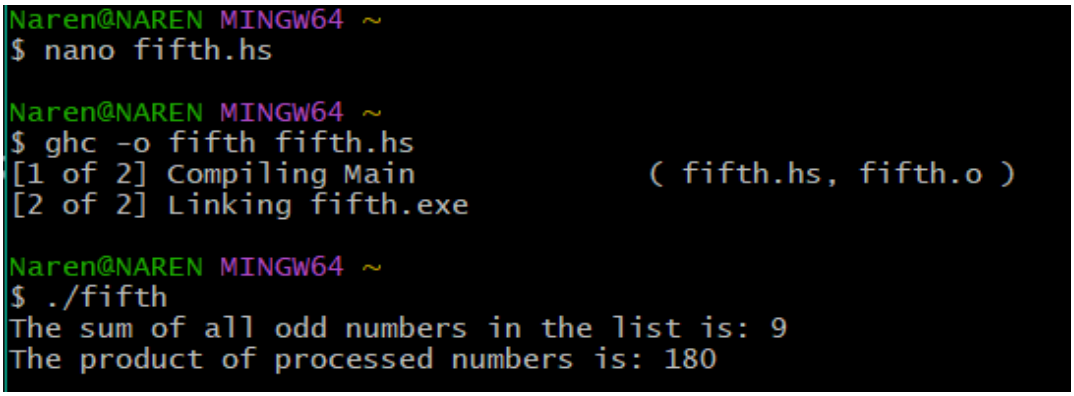
- **`filterAndFold`**: Filters the list with `filterFn` and folds the result using `foldFn`.
- **`sumOfOdds`**: Uses `filterAndFold` to compute the sum of all odd numbers in a list.
- **`main`**: Demonstrates this by calculating the sum of odd numbers in `[1, 2, 3, 4, 5, 6]`, resulting in 9.

5. Map, Filter, and Fold Combination

Problem: Write a function that filters out all numbers greater than 10 from the list [5, 12, 9, 20, 15], doubles each of the remaining numbers, and computes the product of these doubled numbers using foldl.

Code and Output Examples:





Explaination:

The `processNumbers` function:

- Filters numbers ≤ 10.
- Doubles the remaining numbers.
- Calculates their product using foldl.

In the list [5, 12, 9, 20, 15]:

- After filtering: `[5, 9]`.
- After doubling: `[10, 18]`.
- Product: `10 * 18 = 180`.

6. Currying, Map, and Filter

Problem: Write a curried function filterAndMap that takes a filtering function, a mapping function, and a list. It should first filter the list using the filtering function, then apply the mapping function to the filtered elements. Use this function to filter all even numbers from the list [1, 2, 3, 4, 5, 6], double them, and return the result.

Code and Output Examples:

```
~
  GNU nano 8.2                                                    sixth.hs
-- | A curried function that filters and folds a list.
filterAndFold :: (a -> Bool) -> (b -> a -> b) -> b -> [a] -> b
filterAndFold filterFn foldFn initial list = foldl foldFn initial (filter filterFn list)

-- | Function to compute the sum of all odd numbers in a list.
sumOfOdds :: [Int] -> Int
sumOfOdds = filterAndFold odd (+) 0

-- | Function that filters numbers > 10, doubles them, and computes their product.
processNumbers :: [Int] -> Int
processNumbers = foldl (*) 1 . map (*2) . filter (<=10)

-- | A curried function that filters and maps a list.
filterAndMap :: (a -> Bool) -> (a -> b) -> [a] -> [b]
filterAndMap filterFn mapFn = map mapFn . filter filterFn

-- | Main function to demonstrate the usage.
main :: IO ()
main = do
    let numbers = [1, 2, 3, 4, 5, 6]
    let result = sumOfOdds numbers
    putStrLn $ "The sum of all odd numbers in the list is: " ++ show result

    let moreNumbers = [5, 12, 9, 20, 15]
    let productResult = processNumbers moreNumbers
    putStrLn $ "The product of processed numbers is: " ++ show productResult

    let evenNumbersDoubled = filterAndMap even (*2) numbers
    putStrLn $ "The doubled even numbers are: " ++ show evenNumbersDoubled
```

```
Naren@NAREN MINGW64 ~
$ nano sixth.hs

Naren@NAREN MINGW64 ~
$ ghc -o sixth sixth.hs
[1 of 2] Compiling Main             ( sixth.hs, sixth.o )
[2 of 2] Linking sixth.exe

Naren@NAREN MINGW64 ~
$ ./sixth
The sum of all odd numbers in the list is: 9
The product of processed numbers is: 180
The doubled even numbers are: [4,8,12]
```

Explaination:

The `filterAndMap` function:

- Filters a list using a filtering function.
- Applies a mapping function to the filtered elements.

In the list `[1, 2, 3, 4, 5, 6]`:

- Filters even numbers: `[2, 4, 6]`.
- Doubles them: `[4, 8, 12]`.

7. Map, Fold, and Lambda

Problem: Write a function that uses map to convert a list of strings to their lengths, then uses foldl to compute the sum of all string lengths in the list ["hello", "world", "haskell"].

Code and Output Examples:

```
~
  GNU nano 8.2                                                    seven.hs
-- | A curried function that filters and folds a list.
filterAndFold :: (a -> Bool) -> (b -> a -> b) -> b -> [a] -> b
filterAndFold filterFn foldFn initial list = foldl foldFn initial (filter filterFn list)

-- | Function to compute the sum of all odd numbers in a list.
sumOfOdds :: [Int] -> Int
sumOfOdds = filterAndFold odd (+) 0

-- | Function that filters numbers > 10, doubles them, and computes their product.
processNumbers :: [Int] -> Int
processNumbers = foldl (*) 1 . map (*2) . filter (<=10)

-- | A curried function that filters and maps a list.
filterAndMap :: (a -> Bool) -> (a -> b) -> [a] -> [b]
filterAndMap filterFn mapFn = map mapFn . filter filterFn

-- | Function to compute the sum of string lengths in a list.
sumOfStringLengths :: [String] -> Int
sumOfStringLengths = foldl (+) 0 . map (\str -> length str)

-- | Main function to demonstrate the usage.
main :: IO ()
main = do
    let numbers = [1, 2, 3, 4, 5, 6]
    let result = sumOfOdds numbers
    putStrLn $ "The sum of all odd numbers in the list is: " ++ show result

    let moreNumbers = [5, 12, 9, 20, 15]
    let productResult = processNumbers moreNumbers
    putStrLn $ "The product of processed numbers is: " ++ show productResult

    let evenNumbersDoubled = filterAndMap even (*2) numbers
    putStrLn $ "The doubled even numbers are: " ++ show evenNumbersDoubled

    let strings = ["hello", "world", "haskell"]
    let totalLength = sumOfStringLengths strings
    putStrLn $ "The sum of string lengths is: " ++ show totalLength
```

```
Naren@NAREN MINGW64 ~
$ nano seven.hs

Naren@NAREN MINGW64 ~
$ ghc -o seven seven.hs
[1 of 2] Compiling Main             ( seven.hs, seven.o )
[2 of 2] Linking seven.exe

Naren@NAREN MINGW64 ~
$ ./seven
The sum of all odd numbers in the list is: 9
The product of processed numbers is: 180
The doubled even numbers are: [4,8,12]
The sum of string lengths is: 17
```

Explaination:

The `sumOfStringLengths` function:

- Uses `map` with a lambda (`\str -> length str`) to convert strings to their lengths.
- Uses `foldl (+) 0` to sum these lengths.

In the list `["hello", "world", "haskell"]`:

- Lengths: `[5, 5, 7]`.
- Sum: `5 + 5 + 7 = 17`.

8. Filter, Map, and Function Composition

Problem: Define a curried function composeFilterMap that takes a filter function, a map function, and a list. It should first filter the list, then apply the map function to the remaining elements. Use this function to filter out numbers greater than 5 from the list [3, 7, 2, 8, 4, 6], then square the remaining numbers.

Code and Output Examples:

```
~
  GNU nano 8.2                                                    eight.hs
-- | A curried function that filters and folds a list.
filterAndFold :: (a -> Bool) -> (b -> a -> b) -> b -> [a] -> b
filterAndFold filterFn foldFn initial list = foldl foldFn initial (filter filterFn list)

-- | Function to compute the sum of all odd numbers in a list.
sumOfOdds :: [Int] -> Int
sumOfOdds = filterAndFold odd (+) 0

-- | Function that filters numbers > 10, doubles them, and computes their product.
processNumbers :: [Int] -> Int
processNumbers = foldl (*) 1 . map (*2) . filter (<=10)

-- | A curried function that filters and maps a list.
filterAndMap :: (a -> Bool) -> (a -> b) -> [a] -> [b]
filterAndMap filterFn mapFn = map mapFn . filter filterFn

-- | A curried function that filters, maps, and composes operations on a list.
composeFilterMap :: (a -> Bool) -> (a -> b) -> [a] -> [b]
composeFilterMap filterFn mapFn = map mapFn . filter filterFn

-- | Function to compute the sum of string lengths in a list.
sumOfStringLengths :: [String] -> Int
sumOfStringLengths = foldl (+) 0 . map (\str -> length str)

-- | Main function to demonstrate the usage.
main :: IO ()
main = do
    let numbers = [1, 2, 3, 4, 5, 6]
    let result = sumOfOdds numbers
    putStrLn $ "The sum of all odd numbers in the list is: " ++ show result

    let moreNumbers = [5, 12, 9, 20, 15]
    let productResult = processNumbers moreNumbers
    putStrLn $ "The product of processed numbers is: " ++ show productResult

    let evenNumbersDoubled = filterAndMap even (*2) numbers
    putStrLn $ "The doubled even numbers are: " ++ show evenNumbersDoubled

    let strings = ["hello", "world", "haskell"]
    let totalLength = sumOfStringLengths strings
    putStrLn $ "The sum of string lengths is: " ++ show totalLength

    let mixedNumbers = [3, 7, 2, 8, 4, 6]
    let squaredNumbers = composeFilterMap (\x -> x <= 5) (\x -> x * x) mixedNumbers
    putStrLn $ "The squared numbers after filtering are: " ++ show squaredNumbers
```

```
Naren@NAREN MINGW64 ~
$ nano eight.hs

Naren@NAREN MINGW64 ~
$ ghc -o eight eight.h
clang: error: no such file or directory: 'eight.h'
`clang.exe' failed in phase `Linker'. (Exit code: 1)

Naren@NAREN MINGW64 ~
$ ghc -o eight eight.hs
[1 of 2] Compiling Main             ( eight.hs, eight.o )
[2 of 2] Linking eight.exe

Naren@NAREN MINGW64 ~
$ ./eight
The sum of all odd numbers in the list is: 9
The product of processed numbers is: 180
The doubled even numbers are: [4,8,12]
The sum of string lengths is: 17
The squared numbers after filtering are: [9,4,16]
```

Explanation:

The `composeFilterMap` function:

- Filters the list using a filter function.
- Applies a map function to the filtered elements.

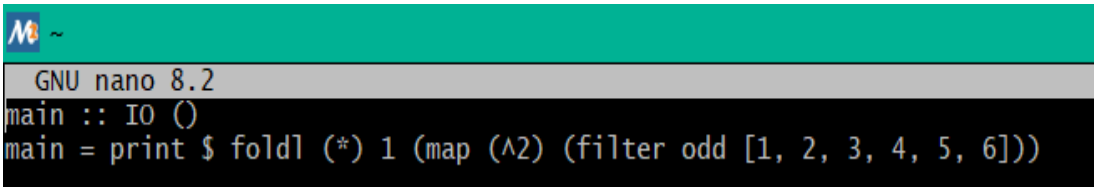
In the list `[3, 7, 2, 8, 4, 6]`:

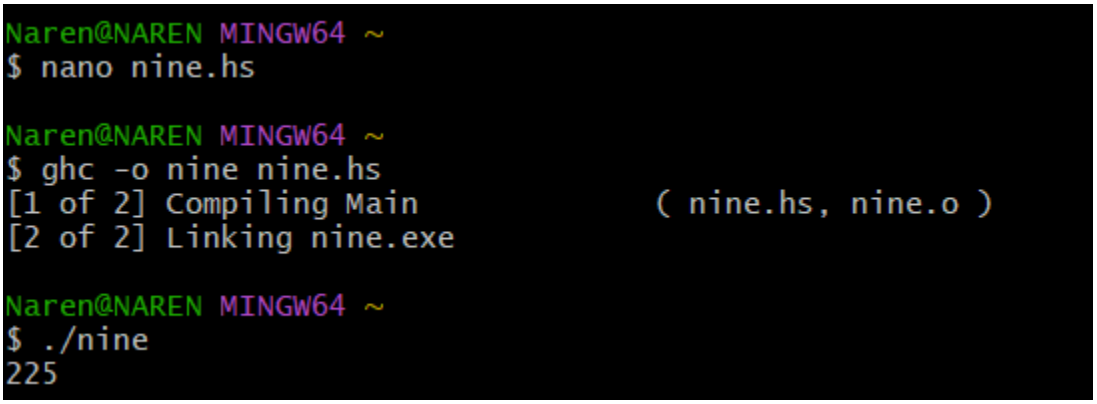
- Filters numbers ≤ 5: `[3, 2, 4]`.
- Squares them: `[9, 4, 16]`.

9. Map, Filter, and Fold Combination

Problem: Use filter to get all odd numbers from the list [1, 2, 3, 4, 5, 6], then square each of these numbers using map, and finally compute the product of the squared numbers using foldl.

Code and Output Examples:





Explanation:

- **filter odd [1, 2, 3, 4, 5, 6]**: This filters out the odd numbers from the list, resulting in `[1, 3, 5]`.
- **map (^2) [1, 3, 5]**: This applies the `square` function (^2) to each element of the filtered list, resulting in `[1, 9, 25]`.

- **foldl (*) 1 [1, 9, 25]**: This uses the `foldl` function to compute the product of the numbers in the list `[1, 9, 25]`. The `(*)` operator is used for multiplication, and the starting accumulator is `1`. The result is `1 * 9 * 25 = 225`.

10. IO Monad and Currying

Problem: Write a program that asks the user for two numbers, then applies a curried function applyOp (which takes an operation and a list) to either sum or multiply the two numbers based on the user's input. First, prompt the user to choose an operation (+ or *), then prompt for the two numbers and return the result of applying the chosen operation.

Code and Output Examples:

```
~
  GNU nano 8.2
import Control.Monad (liftM)

-- Curried function for applying an operation
applyOp :: (Int -> Int -> Int) -> [Int] -> Int
applyOp op [x, y] = op x y

-- Main function to handle IO
main :: IO ()
main = do
    putStrLn "Choose an operation (+ or *):"
    op <- getLine
    putStrLn "Enter the first number:"
    num1 <- liftM read getLine
    putStrLn "Enter the second number:"
    num2 <- liftM read getLine

    let numbers = [num1, num2]
    let result = case op of
                   "+" -> applyOp (+) numbers
                   "*" -> applyOp (*) numbers
                   _   -> error "Invalid operation"
    putStrLn ("The result is: " ++ show result)
```

```
Naren@NAREN MINGW64 ~
$ nano ten.hs

Naren@NAREN MINGW64 ~
$ ghc -o ten ten.hs
[1 of 2] Compiling Main             ( ten.hs, ten.o )
[2 of 2] Linking ten.exe

Naren@NAREN MINGW64 ~
$ ./ten
Choose an operation (+ or *):
+
Enter the first number:
5
Enter the second number:
10
The result is: 15

Naren@NAREN MINGW64 ~
$ ./ten
Choose an operation (+ or *):
*
Enter the first number:
5
Enter the second number:
10
The result is: 50
```

Explaination:

applyOp:

This is a curried function that takes an operation (either addition or multiplication) and a list of two numbers. It applies the operation to the two numbers and returns the result.

Main IO Block:

- First, it prompts the user to choose an operation (+ or *).
- Then, it reads two numbers from the user input.
- The input is processed with `liftM read` to convert the string input into integers.
- Depending on the operation chosen by the user, the program uses the `applyOp` function to either add or multiply the two numbers.
- The result is printed.

Conclusion:

Understanding of core haskell functions, map functions, currying, folding , lambda etc.