Principles of programming languages

Sub Code: 20CYS312 **Name: Chitra Harini**

Roll No:CH.EN.U4CYS22010 **Date: 03-01-2025**

GITHUB LINK: https://github.com/Harini-chitra/Haskell/tree/main/Haskell_Lab5

Lab 5

Exercise 1: Simple Pattern Matching with Integers

Objective: Match integers to return specific string results.

Write a function isZero :: Int -> String that: • Returns "Zero" if the number is 0. • Returns "Not Zero" if the number is anything other than 0.

Haskell Code:

```
isZero :: Int -> String
isZero 0 = "Zero"
isZero _ = "Not Zero"

main :: IO ()
main = do
  putStrLn "Testing isZero:"
  print (isZero 0)  -- Output: "Zero"
  print (isZero 5)  -- Output: "Not Zero"
```

Explanation of code:

- isZero: Uses pattern matching to return a string based on the input integer. If it's 0, it returns "Zero", otherwise it returns "Not Zero".
- main: Tests the isZero function with two example inputs: 0 and 5.

I/O Examples:

```
isZero 0 -- returns "Zero"
isZero 5 -- returns "Not Zero"
```

Output Screenshot:

```
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ gedit one.hs
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ghc one.hs
[1 of 1] Compiling Main             ( one.hs, one.o )
Linking one ...
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ./one
Testing isZero:
"Zero"
"Not Zero"
```

Exercise 2: Pattern Matching on Lists

Objective: Use pattern matching on lists to count the number of elements.

Write a function countElements :: [a] -> Int that returns the number of elements in a list using pattern matching.

Haskell Code:

```
countElements :: [a] -> Int
countElements [] = 0
countElements (_:xs) = 1 + countElements xs

main :: IO ()
main = do
  putStrLn "Testing countElements:"
  print (countElements [1, 2, 3])  -- Output: 3
  print (countElements [])        -- Output: 0
```

Explanation of code:

- countElements: Recursively counts the number of elements in the list. The base case is an empty list [], which returns 0. For non-empty lists, it matches the head element (_) and recursively counts the rest of the list (xs).
- main: Tests countElements with two lists: a list with three elements and an empty list.

I/O Examples:

```
countElements [1, 2, 3] -- returns 3
countElements []        -- returns 0
```

Output Screenshot:

```
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ gedit two.hs
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ghc two.hs
[1 of 1] Compiling Main             ( two.hs, two.o )
Linking two ...
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ./two
Testing countElements:
3
0
```

Exercise 3: Pattern Matching with Tuples

Objective: Matching tuples with simple patterns.

Write a function sumTuple :: (Int, Int) -> Int that takes a tuple of two integers and returns the sum of the integers.

Haskell Code:

```
sumTuple :: (Int, Int) -> Int
sumTuple (x, y) = x + y

main :: IO ()
main = do
  putStrLn "Testing sumTuple:"
  print (sumTuple (3, 5))   -- Output: 8
  print (sumTuple (10, 20)) -- Output: 30
```

Explanation of code:

- sumTuple: Pattern matches the tuple (x, y) and returns the sum of x and y.
- main: Tests the sumTuple function with two different tuples.

I/O Examples:

```
sumTuple (3, 5)   -- returns 8
sumTuple (10, 20) -- returns 30
```

Output Screenshot:

```
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ gedit three.hs
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ghc three.hs
[1 of 1] Compiling Main             ( three.hs, three.o )
Linking three ...
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ./three
Testing sumTuple:
8
30
```

Exercise 4: Pattern Matching on a Custom Data Type

Objective: Define a simple custom data type and pattern match on it.

Define a data type Color to represent basic colors: data Color = Red | Green | Blue Write a function describeColor :: Color -> String that: • Returns "This is Red" if the color is Red. • Returns "This is Green" if the color is Green. • Returns "This is Blue" if the color is Blue.

Haskell Code:

```
data Color = Red | Green | Blue

describeColor :: Color -> String
describeColor Red = "This is Red"
describeColor Green = "This is Green"
```

```
describeColor Blue = "This is Blue"

main :: IO ()
main = do
  putStrLn "Testing describeColor:"
  print (describeColor Red)   -- Output: "This is Red"
  print (describeColor Blue)  -- Output: "This is Blue"
```

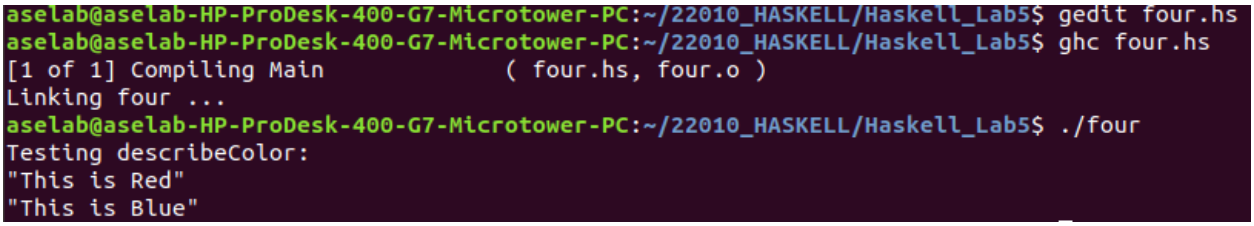
Explanation of code:

- Color: Defines a custom data type with three values: Red, Green, and Blue.
- describeColor: Uses pattern matching to return a string based on the color passed as input.
- main: Tests describeColor with two different colors.

I/O Examples:

```
describeColor Red  -- returns "This is Red"
describeColor Blue -- returns "This is Blue"
```

Output Screenshot:



Exercise 5: Pattern Matching with Lists (Head and Tail)

Objective: Use head-tail pattern matching on lists.

Write a function firstElement :: [a] -> String that returns: • "Empty list" if the list is empty. • "First element is X" if the list has at least one element, where X is the first element.

Haskell Code:

```
firstElement :: Show a => [a] -> String
firstElement [] = "Empty list"
firstElement (x:_) = "First element is " ++ show x

main :: IO ()
main = do
  putStrLn "Testing firstElement:"
  print (firstElement [1, 2, 3])      -- Output: "First element is 1"
  print (firstElement ([] :: [Int]))  -- Output: "Empty list"
```

Explanation of code:

- firstElement: Matches an empty list and returns "Empty list". For a non-empty list, it returns the first element (x).
- main: Tests firstElement with a non-empty list and an empty list.

I/O Examples:

```
firstElement [1, 2, 3] -- returns "First element is 1"
firstElement []        -- returns "Empty list"
```

Output Screenshot:

```
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ gedit five.hs
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ghc five.hs
[1 of 1] Compiling Main             ( five.hs, five.o )
Linking five ...
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ./five
Testing firstElement:
"First element is 1"
"Empty list"
```

Exercise 6: Pattern Matching with Simple List Processing.

Objective: Process lists using pattern matching.

Write a function firstTwoElements :: [a] -> [a] that: • Returns the first two elements of the list if it has two or more elements. • Returns the entire list if it has fewer than two elements.

Haskell Code:

```
firstTwoElements :: [a] -> [a]
firstTwoElements [] = []
firstTwoElements [x] = [x]
firstTwoElements (x:y:_) = [x, y]

main :: IO ()
main = do
  putStrLn "Testing firstTwoElements:"
  print (firstTwoElements [1, 2, 3])     -- Output: [1, 2]
  print (firstTwoElements [10])          -- Output: [10]
  print (firstTwoElements ([] :: [Int]))  -- Output: []
```

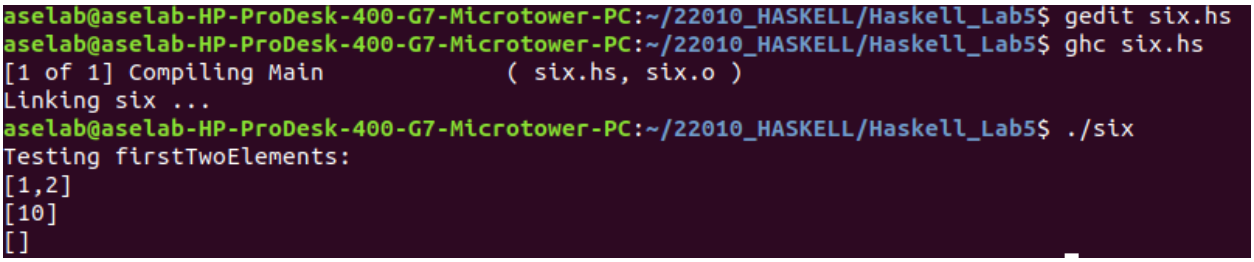
Explanation of code:

- firstTwoElements: Matches a list with at least two elements and returns the first two. If the list has fewer than two elements, it returns the entire list.
- main: Tests firstTwoElements with different list sizes.

I/O Examples:

```
firstTwoElements [1, 2, 3] -- returns [1, 2]
firstTwoElements [10]      -- returns [10]
firstTwoElements []        -- returns []
```

Output Screenshot:



Exercise 7: Pattern Matching with Multiple Cases

Objective: Match against multiple patterns.

Write a function describePair :: (Int, Int) -> String that: • Returns "Origin" if the pair is (0, 0). • Returns "X-Axis" if the first element is 0 and the second element is any non-zero value. • Returns "Y-Axis" if the second element is 0 and the first element is any non-zero value. • Returns "Other" for all other pairs.

Haskell Code:

```
describePair :: (Int, Int) -> String
describePair (0, 0) = "Origin"
describePair (0, _) = "X-Axis"
describePair (_, 0) = "Y-Axis"
describePair _ = "Other"

main :: IO ()
main = do
  putStrLn "Testing describePair:"
  print (describePair (0, 0))  -- Output: "Origin"
  print (describePair (0, 5))  -- Output: "X-Axis"
  print (describePair (3, 0))  -- Output: "Y-Axis"
  print (describePair (2, 3))  -- Output: "Other"
```

Explanation of code:

- describePair: Matches a tuple and returns a description based on the values of the elements in the tuple.
- main: Tests describePair with different pairs of integers.

I/O Examples:

describePair (0, 0) -- returns "Origin"
describePair (0, 5) -- returns "X-Axis"
describePair (3, 0) -- returns "Y-Axis"
describePair (2, 3) -- returns "Other"

Output Screenshot:

```
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ gedit seven.hs
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ghc seven.hs
[1 of 1] Compiling Main             ( seven.hs, seven.o )
Linking seven ...
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ./seven
Testing describePair:
"Origin"
"X-Axis"
"Y-Axis"
"Other"
```

Exercise 8: Pattern Matching for List Recursion.

Objective: Use recursion to work with lists.

Write a function listLength :: [a] -> Int that calculates the length of a list using recursion and pattern matching.

Haskell Code:

```
listLength :: [a] -> Int
listLength [] = 0
listLength (_:xs) = 1 + listLength xs

main :: IO ()
main = do
  putStrLn "Testing listLength:"
  print (listLength [1, 2, 3])  -- Output: 3
  print (listLength [])        -- Output: 0
```

Explanation of code:

- listLength: Recursively counts the number of elements in the list. The base case is an empty list, which returns 0. For non-empty lists, it counts the head (_) and recursively processes the tail (xs).
- main: Tests listLength with a list and an empty list.

I/O Examples:

```
listLength [1, 2, 3] -- returns 3
listLength []        -- returns 0
```

Output Screenshot:

```
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ gedit eight.hs
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ghc eight.hs
[1 of 1] Compiling Main             ( eight.hs, eight.o )
Linking eight ...
aselab@aselab-HP-ProDesk-400-G7-Microtower-PC:~/22010_HASKELL/Haskell_Lab5$ ./eight
Testing listLength:
3
0
```