

20CYS312 - Principles of Programming Languages

S. Shyam Balaji

CH.EN.U4CYS22045

DATE:

20/12/24

Lab Exercise Submission

Git hub link : <https://github.com/shyam150801/principle-of-programming>

Objective of the Exercise:

The goal of this lab is to learn and practice basic pattern matching in Haskell through simple functions. This includes working with integers, lists, tuples, custom data types, and recursive operations.

Exercise 1: Simple Pattern Matching with Integers

Objective:

To write a function that matches an integer to determine if it is zero or not.

Program Code:

```
-- Function to check if a number is zero
isZero :: Int -> String
isZero 0 = "Zero"
isZero _ = "Not Zero"
```

Explanation of the Code:

- Pattern matching is used to check if the input integer is `0`.
- If the number matches `0`, the function returns "Zero".
- The wildcard `_` matches all other numbers, returning "Not Zero".

Input/Output Examples:

```
isZero 0      -- Output: "Zero"
isZero 5      -- Output: "Not Zero"
```

```
asecomputerlab@shyam22045:~$ nano zero.hs
asecomputerlab@shyam22045:~$ cat zero.hs
isZero :: Int -> String
isZero 0 = "Zero"
isZero _ = "Not Zero"

main :: IO ()
main = do
    print (isZero 0)
    print (isZero 5)

asecomputerlab@shyam22045:~$ ghc zero.hs
[1 of 1] Compiling Main                ( zero.hs, zero.o )
Linking zero ...
asecomputerlab@shyam22045:~$ ./zero
"Zero"
"Not Zero"
asecomputerlab@shyam22045:~$ █
```

Exercise 2: Pattern Matching on Lists

Objective:

To count the number of elements in a list using pattern matching.

Program Code:

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

main :: IO ()
```

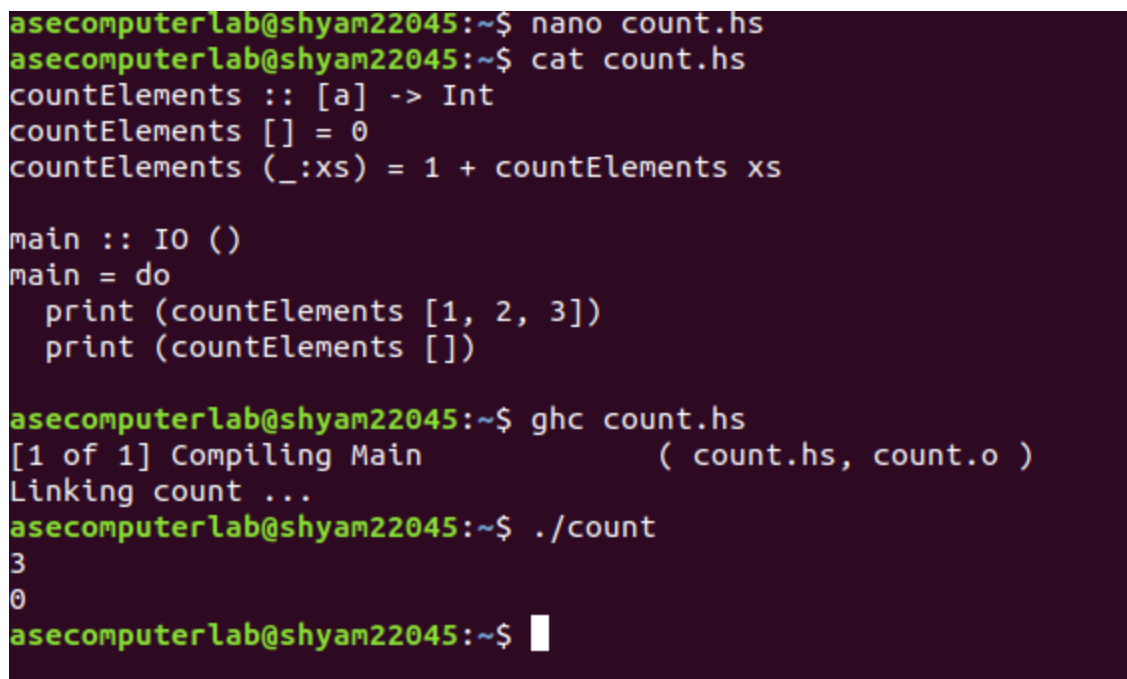
```
main = do
  print (countElements [1, 2, 3])
  print (countElements [])
```

Explanation of the Code:

- The base case matches an empty list `[]` and returns `0`.
- For non-empty lists, the head is ignored (`_`), and the tail (`xs`) is recursively passed to the function, incrementing the count by `1` for each element.

Input/Output Examples:

```
countElements [1, 2, 3]  -- Output: 3
countElements []         -- Output: 0
```



```
asecomputerlab@shyam22045:~$ nano count.hs
asecomputerlab@shyam22045:~$ cat count.hs
countElements :: [a] -> Int
countElements [] = 0
countElements (_:xs) = 1 + countElements xs

main :: IO ()
main = do
  print (countElements [1, 2, 3])
  print (countElements [])

asecomputerlab@shyam22045:~$ ghc count.hs
[1 of 1] Compiling Main                ( count.hs, count.o )
Linking count ...
asecomputerlab@shyam22045:~$ ./count
3
0
asecomputerlab@shyam22045:~$
```

Exercise 3: Pattern Matching with Tuples

Objective:

To calculate the sum of two integers in a tuple.

Program Code:

```
sumTuple :: (Int, Int) -> Int
sumTuple (a, b) = a + b

main :: IO ()
main = do
    print (sumTuple (3, 5))
    print (sumTuple (10, 20))
```

Explanation of the Code:

- The function deconstructs the tuple `(a, b)` and computes the sum of its two elements.

Input/Output Examples:

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

```
asecomputerlab@shyam22045:~$ nano tuple.hs
asecomputerlab@shyam22045:~$ cat tuple.hs
sumTuple :: (Int, Int) -> Int
sumTuple (a, b) = a + b

main :: IO ()
main = do
    print (sumTuple (3, 5))
    print (sumTuple (10, 20))

asecomputerlab@shyam22045:~$ ghc tuple.hs
[1 of 1] Compiling Main                ( tuple.hs, tuple.o )
Linking tuple ...
asecomputerlab@shyam22045:~$ ./tuple
8
30
asecomputerlab@shyam22045:~$
```

Exercise 4: Pattern Matching on a Custom Data Type

Objective:

To define a custom data type and describe colors using pattern matching.

Program Code:

```
-- Custom data type for colors
data Color = Red | Green | Blue

-- Function to describe a color
describeColor :: Color -> String
describeColor Red    = "This is Red"
describeColor Green  = "This is Green"
describeColor Blue   = "This is Blue"
```

Explanation of the Code:

- A data type `Color` is defined with three possible values: `Red`, `Green`, and `Blue`.
- The function matches each value and returns the corresponding description.

Input/Output Examples:

```
describeColor Red    -- Output: "This is Red"
describeColor Blue   -- Output: "This is Blue"
```

```
asecomputerlab@shyam22045:~$ nano color.hs
asecomputerlab@shyam22045:~$ cat color.hs
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
    print (describeColor Red)
    print (describeColor Blue)

asecomputerlab@shyam22045:~$ ghc color.hs
[1 of 1] Compiling Main                ( color.hs, color.o )
Linking color ...
asecomputerlab@shyam22045:~$ ./color
"This is Red"
"This is Blue"
asecomputerlab@shyam22045:~$
```

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

Objective:

To retrieve the first element of a list or return a message if the list is empty.


Program Code:

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

main :: IO ()
```

```
main = do
  print (firstElement [1, 2, 3]) -- Example with integers
  print (firstElement ["hello", "world"]) -- Example with strings
  print (firstElement []) -- Empty list case
```

Explanation of the Code:

- The base case matches an empty list and returns "Empty list".
- For non-empty lists, the head () is extracted and concatenated into the output string.

Input/Output Examples:

```
"First element is 1"
"First element is \"hello\""
"Empty list"
"Empty list"
```

```

asecomputerlab@shyam22045:~$ nano head.hs
asecomputerlab@shyam22045:~$ cat head.hs
firstElement :: Show a => [a] -> String
firstElement []      = "Empty list"
firstElement (x:_) = "First element is " ++ show x

main :: IO ()
main = do
    print (firstElement [1, 2, 3] :: String) -- List of integers
    print (firstElement ["hello", "world"] :: String) -- List of strings
    print (firstElement ([] :: [Int])) -- Empty list of integers
    print (firstElement ([] :: [String])) -- Empty list of strings

asecomputerlab@shyam22045:~$ ghc head.hs
[1 of 1] Compiling Main                ( head.hs, head.o )
Linking head ...
asecomputerlab@shyam22045:~$ ./head
"First element is 1"
"First element is \"hello\""
"Empty list"
"Empty list"
asecomputerlab@shyam22045:~$

```

Exercise 6: Pattern Matching with Simple List Processing

Objective:

To return the first two elements of a list or the entire list if it has fewer than two elements.

Program Code:

```

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

main :: IO ()
main = do
    print (firstElement [1, 2, 3])

```



```
print (firstElement ([] :: [Int]))
```

Explanation of the Code:

- If the list has at least two elements, the first two are extracted and returned.
- Otherwise, the function returns the entire list.

Input/Output Examples:

```
First element is 1
Empty list
```

```
asecomputerlab@shyam22045:~$ nano first.hs
asecomputerlab@shyam22045:~$ cat first.hs
firstElement :: Show a => [a] -> String
firstElement [] = "Empty list"
firstElement (x:_) = "First element is " ++ show x

main :: IO ()
main = do
    print (firstElement [1, 2, 3])
    print (firstElement ([] :: [Int])) -- Explicitly specify the type of the empty list

asecomputerlab@shyam22045:~$ ghc first.hs
[1 of 1] Compiling Main             ( first.hs, first.o )
Linking first ...
asecomputerlab@shyam22045:~$ ./first
"First element is 1"
"Empty list"
asecomputerlab@shyam22045:~$
```

Exercise 7: Pattern Matching with Multiple Cases

Objective:

To describe a pair of integers based on specific conditions.

Program Code:

```
describePair :: (Int, Int) -> String
describePair (0, 0) = "Origin"
```

```
describePair (0, _) = "X-Axis"
describePair (_, 0) = "Y-Axis"
describePair _      = "Other"

main :: IO ()
main = do
    print (describePair (0, 0))
    print (describePair (0, 5))
    print (describePair (3, 0))
    print (describePair (2, 3))
```

Explanation of the Code:

- The function checks multiple conditions on the tuple and returns the appropriate description.

Input/Output Examples:

```
(describePair (0, 0))
(describePair (0, 5))
(describePair (3, 0))
(describePair (2, 3))
```

```

asecomputerlab@shyam22045:~$ nano pari.hs
asecomputerlab@shyam22045:~$ ghc pari.hs
[1 of 1] Compiling Main                ( pari.hs, pari.o )
Linking pari ...
asecomputerlab@shyam22045:~$ cat pari.hs
describePair :: (Int, Int) -> String
describePair (0, 0) = "Origin"
describePair (0, _) = "X-Axis"
describePair (_, 0) = "Y-Axis"
describePair _     = "Other"

main :: IO ()
main = do
    print (describePair (0, 0))
    print (describePair (0, 5))
    print (describePair (3, 0))
    print (describePair (2, 3))

asecomputerlab@shyam22045:~$ ./pari
"Origin"
"X-Axis"
"Y-Axis"
"Other"
asecomputerlab@shyam22045:~$ █

```

Exercise 8: Pattern Matching for List Recursion

Objective:

To calculate the length of a list recursively.

Program Code:

```

-- Function to calculate list length
listLength :: [a] -> Int
listLength []      = 0
listLength (_:xs) = 1 + listLength xs

```

Explanation of the Code:

- The base case matches an empty list and returns 0.

- For non-empty lists, the head is ignored, and the tail is recursively passed to compute the length.

Input/Output Examples:

```
listLength [1, 2, 3]  -- Output: 3
listLength []         -- Output: 0
```

```
asecomputerlab@shyam22045:~$ nano length.hs
asecomputerlab@shyam22045:~$ cat length.hs
listLength :: [a] -> Int
listLength [] = 0
listLength (_:xs) = 1 + listLength xs

main :: IO ()
main = do
    print (listLength [1, 2, 3])
    print (listLength [])

asecomputerlab@shyam22045:~$ ghc length.hs
[1 of 1] Compiling Main                ( length.hs, length.o )
Linking length ...
asecomputerlab@shyam22045:~$ ./length
3
0
asecomputerlab@shyam22045:~$
```

Conclusion:

This lab exercise provided hands-on experience with pattern matching in Haskell. We practiced matching integers, lists, tuples, and custom data types, and applied recursion for list processing.
