Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Laboratory 7

Title of the Laboratory Exercise: Types, Tuples, Lists and Strings and Pattern Matching

1. Introduction and Purpose of Experiment

   Students apply various basic data types and perform simple operations using patterns in Haskell

2. Aim and Objectives

   Aim

   - To apply correct data types and patterns for solving problems with functional programming

   Objectives

   At the end of this lab, the student will be able to

   - Apply correct data type for given problem
   - Apply pattern matching for function definition
   - Solve problems using functional paradigm
   - Manipulate lists

3. Experimental Procedure

For the problems listed below, design the data structures, algorithm(s) and write the program(s). Tabulate the output for various inputs and verify against expected values. Compare the programming method in Haskell with C programming language. Describe your learning along with the limitations of overall approach if any. Suggest how these can be overcome.

a. Find area of a triangle using Tuples and functions in Haskell

b. Filter only capital letters in a given string using functions and lists using Haskell

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Documentation:	
a. Procedure and Algorithm(s):	**Procedure:** 1. Creating a new Haskell File Haskell source files are ASCII based text file with a `.hs` extension. The execution of the Haskell Project usually begins from `Main.hs` and hence we name the file Main.hs, it can be anything if its going to a simple project, for the sake of organization the file that contains the driver function or main function is inside the Main.hs. 2. Design the functions required Since Haskell is a functional programming language, the required functions need to be decomposed into sub functions, the operations that needs to be done in the function also needs to be purely functional, and side-effects needs to be avoided as much as possible. 3. Documentation Write documentation for the functions that are used in the source file along with its input and outputs, write the function signatures to enforce typing in Haskell. 4. Running and testing The functions written can be individually tested in GHCI, which is an interactive version of the Glasgow Haskell Compiler, this will us know test atomic functions, which are a part of more functions, and make it easier to debug the programs, since there are no side-effects testing becomes easier. Load the `.hs` file into GHCI by typing `:l Main.hs` , this will compile and load the function written in the file and each of the function can be called specifically by just typing the name of the function, to see the type of function `:t` can be used.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

	<u>Algorithms:</u>
	areaOfTriangle
	Params: (Base, Height)
	`Step 1: Start`
	`Step 2: return 0.5 * Base * Height`
	`Step 3: Stop`
	
	filterCapital
	Params: String S
	`Step 1: Start`
	`Step 2: Take every element from S such that the element belong to the list ['A'..'Z']`
	`Step 3: Stop`

b. Conclusions :	The knowledge of usage of tuples, functions and list comprehensions was learned in this experiment. HUnit testing can be used by software developers to test and debug Haskell code, since we have already done functional decomposition it is easy to debug such functions.

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

Results and Discussions:

Screenshot:

```
areaOfTriangle :: Fractional a => (a, a) -> a
areaOfTriangle (b, h) = 0.5 * b * h
```

OUTPUT :

```
*Main> areaOfTriangle (1,2)
1.0
*Main> areaOfTriangle (5,1)
2.5
*Main> areaOfTriangle (1000,1245)
622500.0
*Main> areaOfTriangle (1287398314,321984723847)
2.0726129530719168e20
```

Discussion:

To compute the area of the triangle we need the base and height of the triangle, these are the two input arguments to the function, and it returns the area of the triangle. Since all of these can be fractional values, we put the type constraint `Fractional` to the input arguments as well as the output value.

The argument is a tuple of the base and the height. The simple area of triangle formula is used and the area of that triangle is returned. Another way to solve this would be to take the argument tuple as `x` and to get the base and height, `fst` and `snd` functions can be used.

Screenshot:

```
filterCapital :: String -> String
filterCapital s = [c | c <- s, c `elem` ['A'..'Z']]
```

OUTPUT:

```
*Main> filterCapital "IdontLIKEFROGS"
"ILIKEFROGS"
*Main> filterCapital "AaBbCcDdEeFf"
"ABCDEF"
*Main> filterCapital "abcdef"
""
*Main> filterCapital "ABCDEF"
"ABCDEF"
```

Discussion:

Name: SATYAJIT GHANA Registration Number: 17ETCS002159

`filterCapital` is function that takes a String as an argument and returns another String as the result. The aim is to filter only the upper-case letters, so we use list comprehension with the predicate that the character in the string in an upper-case. Given a string s, using the generator (<-) take every character of s, and apply the predicate `c `elem` ['A'..'Z'],elem` is a function that takes in two arguments, a element and a list of elements, and returns if that element is present in that list. We use list comprehension to generate the list of upper-case letters, and the predicate here checks if that character is present in this list or not. This filters only the upper-case letters from the given string.