**TCSS 380, Spring 2024, Lab 1 (Part 1)**

The objective of this lab assignment is to illustrate in Erlang some of the textbook concepts that appear in chapter 15: basic syntax of a functional language.

You should work with a team of three to complete this lab according to the instructions provided via Canvas. You should have all the required software installed and tested before starting this lab. Each file that you submit is to have all team members names listed at the top of it. Submit only once per partnership. You must complete the entirety of the lab as a team. Outside of class, it is OK to use synchronous remote video/screen share collaboration.

Altogether, you need to upload into Canvas:

• lab01.erl  
• screenshot/s of running lab01.erl in the shell showing the function calls (png, jpg, or gif)

In the text editor of your choosing create a file **lab01.erl** and write function definitions that could be run from the shell – make sure you use the exact same names (anything in italics) as indicated in function descriptions. Pay attention to directions that specify what syntactical mechanisms to use within the functions (the purpose of this lab is to make sure you are familiar with most of the basic Erlang syntax discussed this week).

1. Function *velocity* that takes two arguments (d and t) representing distance and time and returns the velocity of an object using the following equation:

**V = d / t**

1. assume that all inputs are positive numbers.
2. you do NOT have to guard against invalid input
3. you may NOT use: functions defined in the math library

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *isDifferent* that takes two arguments and returns **false** if these two values are the same considering both the value and the type and **false** otherwise. For example, if 5.0 and 5 are passed, the function should return **true** because these two parameters have different data types; if 5.0 and 4.0 are passed, the function should return **true** because these two parameters have different values; if 5 and 5 are passed, the function should return **false** because these two parameters have the same value and data type.
   1. This should be a one-liner!

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *parity* that takes an argument and returns an **atom** (***even***)if the argument is divisible by 2, and an **atom** (***odd***) if the argument is not divisible by 2. Use function level pattern matching (several function clauses / headers with guards) instead of an *if* expression. Assume an integer will be passed, i.e. there is no need to handle non-integer values.
   1. Simplify! Use as little “logic” as possible. Attempt direct matches.
      1. This should be accomplished with two function clauses.
      2. This should be accomplished with a singular guard on one of the function clauses.
   2. assume that all inputs are numbers
   3. you do NOT have to guard against invalid input

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *colorOfTheDay* that takes an **atom** argument and returns its equivalent color as an **atom**. Use function level pattern matching (several function clauses / headers) instead of a case expression inside a function. Use the following values:
   1. for value *monday* return *red*
   2. for value *tuesday*  returns *orange*
   3. for value *wednesday* returns *yellow*
   4. for value *thursday* returns *green*
   5. for value *friday* returns *blue*
   6. for value *saturday* returns *indigo*
   7. for value *sunday* returns *violet*
   8. for an invalid value / type, returns *no\_match*

✓ Run the function in the Erlang shell and take and save a screenshot

1. TWO functions *myPerfectSqrt* that implements the recursive math formula provided below – do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching

Function 1:

myPerfectSqrt(m) -> myPerfectSqrt(m, 1, 0).

Function 2:

|  |  |  |  |
| --- | --- | --- | --- |
| myPerfectSqrt (m, n, a) = | page2image23339648 | a | m == 0 |
| myPerfectSqrt (m - n, n + 2, 1 + a) | m > 0 |
| the atom: **not\_perfect** | m < 0 |

When testing this function, myPerfectSqrt *(4)* should result in 2, myPerfectSqrt *(16)* should result in 4, myPerfectSqrt *(17)* should result in **not\_perfect**.

* 1. Do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching
  2. assume that all inputs are numbers
  3. assume all inputs are positive
  4. you do NOT have to guard against invalid input

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *mySum* that implements the following Java code as recursion in Erlang:

**int sum ( int boundary ) {**

**int sum = 0;**

**for(int i = 1; i <= boundary; i++)**

**sum += i;**

**return sum;**

**}**

* 1. Do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching
  2. assume that all inputs are numbers
  3. you do NOT have to guard against invalid input

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *divide* that takes two integer arguments and returns a single integer number. This function should result in division of the first argument by the second argument, however, you are only allowed to use the – (subtraction) arithmetic operator. ***You may not use any other arithmetic operators nor functions from any other module****.* The return should be an integer representing the quotient of this division.
   1. Do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching
   2. assume that all inputs are *positive* numbers
   3. Given arguments A and B, the following are all valid:
      1. A > B
      2. A = B
      3. A < B
   4. you do NOT have to guard against invalid input

✓ Run the function in the Erlang shell and take and save a screenshot

1. **BONUS CHALLENGE PROBLEM:** Function *fullDivide* that takes two integer arguments and returns a tuple. This function should result in division of the first argument by the second argument, however, you are only allowed to use the – (subtraction) arithmetic operator. ***You may not use any other arithmetic operators nor functions from any other module****.* The first element in the tuple should be a number representing the quotient and the second element a number representing the remainder.
   1. Do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching
   2. assume that all inputs are *positive* numbers
   3. Given arguments A and B, the following are all valid:
      1. A > B
      2. A = B
      3. A < B
   4. you do NOT have to guard against invalid input
   5. I was forced to implement this solution using *tail recursion*. I’m not sure it can be implemented using non-tail recursion.

✓ Run the function in the Erlang shell and take and save a screenshot

1. Function *labStatistics* that includes SIDE EFFECTS (oh no!). This function should accept a single tuple argument. The tuple should include three elements in the following order:

* One of the two atoms: leader | student
* A string that represents a Name
* A floating point number [0.0 – 1.0] that represents the students contribution to the lab

Using one call to one of the output functions in the io module, print to the terminal the following string based on the tuple.

* When the first element is the atom leader:
  + “LEADER: <Insert the value for Name>’s contribution was <Insert the value for contribution – as a percentage><Newline>”
* When the first element is the atom student:
  + “<Tab><Insert the value for Name>’s contribution was <Insert the value for contribution – as a percentage><Newline>”

Upon completion of the output function used from the io module, this function should evaluate to the atom leader | student dependent on the value in the argument.

* 1. Do NOT write any if statements or case expressions inside the function body but rather use function level pattern matching
  2. you do NOT have to guard against invalid input
  3. Research accepting a tuple as an argument to a function and how to decompose it
     1. <https://learnyousomeerlang.com/syntax-in-functions#pattern-matching>
  4. Research the different functions in the io module and how to properly use the format specifiers.
     1. <https://www.erlang.org/doc/man/io>

1. Function *labOne* that makes calls to the labStatistics function. For each student in the group, make an individual call to the labStatistics function accurately describing the students role and contribution to this lab. (You will hard-code the literal values into the tuples used in this function) This function should evaluate to the atom done.

✓ Run the function in the Erlang shell and take and save a screenshot

**EOF**