# CS431: Programming Languages Lab (2020) Assignment 3(Functional Programming)

#### Name - Rishi Pathak

#### Roll Number - 170101054

**Problem 3: House Planner:**

1. *Write the algorithm (in pseudo-code) that you devised to solve the problem*
2. 1. Find and store all the possible dimensions of the rooms. I.e.- (10x10) to (15x15) will contain (10x11), (10x12)…. etc.
3. Set the number of rooms based on the number of Hall and Bedroom count
4. Make list1 which is list of unique tuple of dimensions filtering tuples with same total area (bedroom, hall).
5. Similarly make list2, list3, list4 and list5 to get set of dimensions of rooms using previous list and set of dimensions of next room.
6. Next find the maximum area possible of the set of dimensions in the list5.
7. Find the dimensions of room corresponding to max area.
8. Print the results.

*Q) How many functions did you use?*

A) I used 20 functions. 5 for making Cartesian product set of dimensions of rooms. 10 functions for making the sets unique, 2 functions for finding all possible dimensions of the room, 2 for finding max area and results, and 1 main design function to combine them all.

*Q) Are all those pure?*

A) Yes all of my functions are pure. That is no matter how many times we call the functions with same arguments, the functions will return the same value. Haskell function “getLine” is an impure function, which I ignored using deliberately because it can return different values each time it is called.

*Q) If not, why? (Means, why the problem can’t be solved with pure functions only).*

A) It is because Haskell makes huge attempts to separate “pure” code from “impure” code. In the context of an IO action like getLine, this translates into “encapsulating” a value resulting from an IO into a “monad” — in that sense, a “monad” is like a box in which results of an IO is either put in, or extracted from. The function getLine returns an IO String: it gets a line from the outside world, encapsulates in the monad type IO String, and then returns back.

*Q) Do you think the lazy evaluation feature of Haskell can be exploited for better*

*performance in the solutions to the assignments? If so, which solution(s) and how?*

A) Yes, the lazy evaluation of Haskell is very useful as it doesn’t calculates value of a function is called until necessary. It greatly improves the performance of the program. In the solution of Question 3, I made many functions which were called by other functions. Like in making the list of set of dimensions of the room, makeset functions don’t evaluate and are passed to the makeunique functions as it is and then later evaluated when filter is applied.

*Q) We can solve the problems using any imperative language as well. Do you find any*

*advantage of using Haskell for these problems (w.r.t the property of lack of side effect)? If*

*your answer is no, elaborate on why not?*

A) Removing side-effects from the equation allows expressions to be evaluated in any order. A function will always return the same result if passed the same input. This has no exceptions. This determinism removes a whole class of bugs found in imperative programs. In fact, most bugs in large systems can be traced back to side-effects; if not directly caused by them, then caused by a flawed design that relies on side-effects. This means that functional programs tend to have far fewer bugs than imperative ones.