CS-431 | PROGRAMMING LANGUAGES LAB REPORT
ASSIGNMENT-3
VAKUL GUPTA | 170101076

NOTE- Each Problem Folder contains the corresponding **README** file. Check that for compilation and execution.

PROBLEM 3 - House Planner

a) Write the algorithm (in pseudo-code) that you devised to solve the problem (you must not write the code for the algorithm in the report).

- ➔ We take the total area, the number of bedrooms, and the number of Halls as the input.
- ➔ Compute all the possible dimensions for each component of the room with the given range (x1, y1) to (x2, y2).
- ➔ Compute the number of Kitchens and Bathrooms based on the input given.
- ➔ Then we get those combinations of Bedrooms, halls, kitchen, bathrooms, balcony, and garden by concatenating each dimension tuple of all components.
- ➔ Also, we filter those combinations that have dimensions of kitchen less than that of hall and bedrooms both.
- ➔ Also, we choose those combinations where the bathroom has dimensions less than that of the kitchen.
- ➔ We remove unwanted tuples that have the same area, as this won't affect the final answer.
- ➔ Then out of all the feasible combinations, we find the maximum area of all the tuples.
- ➔ We find that tuple (bedroom, halls, kitchen, bathrooms, balcony, kitchen) which have the area with max feasible area.
- ➔ Print the output in the required format.

b) How many functions did you use?

These are the functions used in the implementation:

```
1. getMaximumArea
2. getOptimalDesign
3. findAllPossibleDimensions
4. findAllPossibleDimensionsUtility
5. twoTupleCombination
6. threeTupleCombination
7. fourTupleCombination
8. fiveTupleCombination
9. sixTupleCombination
```

```
10.   filterUniqueCombFour
11.   filterUniqueCombFourUtility
12.   filterUniqueCombFive
13.   filterUniqueCombFiveUtility
14.   filterUniqueCombSix
15.   filterUniqueCombSixUtility
```

c) Are all those pure?

- ➔ **Definition of Pure Functions** -> In computer programming, a pure function is a function that has the following properties: Its return value is the same for the same arguments (no variation with local static variables, non-local variables, mutable reference arguments or input streams from I/O devices).

- ➔ No, **getOptimalDesign** is an impure function as it uses the I/O method and uses **putStrLn** function which is impure. Rest all the functions used in the implementation of the above problem are pure satisfying all the properties of pure functions mentioned above.

d) If not, why? (Means, why the problem can't be solved with pure functions only).

As the input and output are necessary for our problem, that's why this problem of impure function is occurring in our case. Also, this can't be solved with pure functions as this has to be necessarily done in our case.

IN ADDITION, write short notes on the following in the report.

a) 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?

Lazy evaluation is a method to evaluate a Haskell program. It means that expressions are not evaluated when they are bound to variables, but their evaluation is deferred until their results are needed by other computations. In consequence, arguments are not evaluated before they are passed to a function, but only when their values are actually used.

Yes, the lazy evaluation feature of Haskell can be exploited for better performance in the solutions to the assignments, because it helps in handling lists of very big size as they are

evaluated only when needed. In **Problem-3**, the computation of all possible dimensions for the rooms is not executed when calling the function but when it is needed.
Hence, lazy evaluation saves computation time and memory.

b) 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?

Calling a function once is the same as calling it twice and discarding the result of the first call. In fact, if you discard the result of any function call, Haskell will spare itself the trouble and will never call the function.
Haskell has a huge advantage because of the **lack of side effect**. Lack of side effect means that variables are not changed once declared, and hence we can use parallel processing to compute such variables as they will not be changed at any point of execution. In such cases, we do not need to worry about data consistency. Because of this reason, there won't be many critical sections and hence we can have parallel processing to save computational time.
