CS431: Functional Programming Assignment

Name: Arpit Gupta
Roll No.: 170101012

Q1) Write the algorithm (in pseudo-code) that you devised to solve the problem?

Given total available space, number of bedrooms and number of halls we have to find the optimum number of different types of rooms and their sizes so as to minimize the unused space. The constraints given for the problem are:

1) Dimension Kitchen <= Dimension Hall
2) Dimension Kitchen <= Dimension Bedroom
3) Dimension Kitchen > Dimension Bathroom
4) Number of kitchens is one third of number of bedrooms
5) Number of bathrooms = number of bedrooms + 1
6) Number of garden and balcony is 1 each

The pseudo code is

```
design maxAreaPossible numBedrooms numHalls
{
    Compute the count of different types of rooms using constraints given
    // Tuple generator:
        1) find tuples of dimensions of bedroom, hall, kitchen, bathroom which give
        different area i.e. somewhat like a set
        2) Also checks for constraints for each tuple

    b_h_k_ba_tuples = Tuple generator (maxAreaPossible,numBedrooms,numHalls,
                                       numKitchen numBathroom,
                                       bedroomHallKitchenDimensionTuple)

    // find tuples of dimensions of bedroom, hall, kitchen, bathroom, garden which give
    different areas
    b_h_k_ba_g_tuples = Tuple generator (maxAreaPossible,numBedrooms,numHalls,
                                         numKitchen, numBathroom,
                                         numGarden,b_h_k_ba_tuples)
    // find tuples of dimensions of bedroom, hall, kitchen, bathroom, garden, balcony which
    give different areas
b_h_k_ba_g_bal_tuples = Tuple generator (maxAreaPossible,numBedrooms,numHalls,
```

```
                                        numKitchen, numBathroom,
                                        numGarden,numBalcony,b_h_k_ba_g_tuples)

    // Now we have tuples which have dimensions of all the rooms and which satisfy the
    constraints given above

    // Compute the maximum possible area from tuples obtained
    maxArea = getMaxArea(b_h_k_ba_g_bal_tuples,numBedrooms,numHalls,
                                    numKitchen, numBathroom, numGarden,numBalcony)

    // Get final dimensions of rooms based on max area we obtained above
    finalDimensions = getDimension (maxArea, b_h_k_ba_g_bal_tuples,numBedrooms,
                                        numHalls,numKitchen, numBathroom,
                                        numGarden,numBalcony)

    // Since finalDimension can have multiple answers we just take the first one and print it
    printOptimumAns(finalDimension[0], numBedrooms,
                        numHalls,numKitchen, numBathroom, numGarden,numBalcony)
}
```

Q2) How many functions did you use?

Functions used are:

- design
- computeTuplesof4
- computeTuplesof5
- computeTuplesof6
- b_h_k_ba_Generator
- b_h_k_ba_g_Generator
- b_h_k_ba_g_bal_Generator
- getMaximumArea
- computeArea
- printOptimumAnswer

Above functions also some util functions:

- b_h_k_ba_GeneratorUtil
- b_h_k_ba_g_GeneratorUtil
- b_h_k_ba_g_bal_GeneratorUtil
- getMaximumAreaUtil
- condition1
- condition2

Total number of functions used are **16**. Some procedures like *possibleDimensionBedroom* are not counted as functions as they simply provide lists of dimensions.

Q3) Are all those pure?

A pure function is defined as the one which gives the same output on the same input (passed as argument) and also has no side effects on the outside world i.e. no state change in haskell. For above reasons IO operations are not considered as pure functions. So **design** and **printOptimumAnswer** are the only impure functions in my program.

Q4) If not, why? (Means, why the problem can't be solved with pure functions only).

Since printOptimumAnswer prints the final output and the design function calls it, they are both impure functions. The problem can't be solved with only pure functions because we have to print the final answer in specified format. Had the output not been asked in specified format but only as a list or some other data type then I would have used only pure functions.

Short Notes

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 very powerful feature of haskell. It means that any expression which is bound by any variable is not calculated until it is absolutely needed by the program to be evaluated. Such type of evaluation is useful if we need to define large variables, like a list of 12,000 elements of integers, which would take about 48KB of space. Such variables will not be assigned any space in memory until it is called by any function.

In the assignment I used lazy evaluation to good effect in question 2. In question 2 to generate random fixtures I have taken a different permutation of teams list on the basis of seed (generated by a random number generator), now there are a total of 12! permutations of teams list, but since we need only the permutation indexed by seed so haskell uses lazy evaluation to find permutations only upto seed thus saving a lot of 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?

Haskell has many advantages over conventional imperative languages like purity, lazy evaluation and strong typing. One of the main features of haskell is **lack of side effects,** this is because haskell is a pure language, which means that results of any function are the same for the same arguments passed. Also haskell functions don't have any effects on the outside world like writing files or sending data over the network. These features of haskell allow us to replace a function call with the results of its previous call with the same parameters. This also allows us to write modular code and makes debugging easy.