CS 431
Programming Languages Lab
Assignment 3

Anubhav Tyagi

170101009

November 28, 2020

Q1 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)

Algorithm 1 Design totalArea numBedroom numHall

$numKitchen \leftarrow (numBedroom + 2)/3$
$numBathroom \leftarrow numBedroom + 1$
$numBalcony \leftarrow 1$
$numGarden \leftarrow 1$

Initialise the minimum dimensions of each room type

searchList1 $\leftarrow$ Generate all possible combinations of bedrooms, halls, kitchen and bathroom s.t. $area(bedroom) * numBedroom + area(hall) * numHall + area(kitchen) * numKitchen + area(bathroom) * numBathroom <= totalArea$

searchList2 $\leftarrow$ Generate all possible combinations of balcony and garden s.t. $area(balcony) + area(garden) <= totalArea$

sortedList1 $\leftarrow$ sort searchList1 in increasing order of area

sortedList2 $\leftarrow$ sort searchList2 in decreasing order of area

outputDesigns $\leftarrow \{(x, y) \mid x\epsilon$ sortedList1, $y\epsilon$ sortedList2, $area(x) + area(y) <= totalArea\}$

optimalPlan $\leftarrow x$ s.t. $x\epsilon$ outputDesigns and $area(x) <= area(y)$ $\forall y\epsilon$ outputDesigns

return optimalPlan

Q2 How many functions did you use?

A total of 8 functions were used in the program named as - design, searchBedroom, searchHall, searchKitchen, searchBathroom, searchBalcony, searchGarden and mergeDesigns. Two other functions were used as custom sort functions to order designs by their area. These two functions are - sortOrderStruct1 and sortOrderStruct2.

Q3 Are all those pure?

Yes, all of these functions are pure. According to the definition of pure functions, if you call pure function with same set of arguments, it will always return the same values. In all of the functions used, the output of the function solely depends on the input provided to the function and the output is same everytime the functions is called with same set of parameter values.Hence, all the functions used are pure

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

The solution implemented uses only pure functions and hence it can be solved using pure functions

Q5 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?

Yes, Haskell's lazy evaluation has many advantages and can be exploited in many ways. First is that, the lazy evaluation allows for declaration of a large number of structures as abstractions instead of primitives. What this means is, whenever we define a *class* or *struct* in C/C++, the compiler reserves space for it even if it is not used in the program. This limits the number of *class* or *struct* that can be defined in the program.However, Haskell only allocates space to structures whenever they are called.The lazy evaluation also saves computation time by skipping computations portions of program that are not used. This speeds up the execution time of the program.Any expression in Haskell is not calculated unless it is absolutely necessary. Even when passing expressions to functions, the expressions are not evaluated until their values are needed in the program.

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?

A function or expression is said to have a side effect if it modifies some state variables outside its local environment. Examples of side effects include modifying a non-local variable, modifying a static local variable, modifying a mutable argument passed by reference or performing IO. Having side effects makes a function unpredictable. It becomes hard to debug the program and predict the output of the function. When a function has no side effects, the programmer can do formal verifications of the program. Since, Haskell does not have any side effects, it makes it easier to debug the program and formally verify the output of the program.