Assignment Set 3 (Functional Programming with Haskell)

- **Assignment will be evaluated by the TAs.**
- **You should submit report (for answering non-code related questions), complete source codes and executable files.**
- **All codes must be properly documented and good code writing practice should be followed (carry marks).**
- **Copying is strictly prohibited. Any case of copying will automatically result in F for the whole course, irrespective of your performance in the other parts of the lab.**

- **Submission deadline: 20 November, 2019**

- **Marks distribution: 10, 10, 30, (40+10)**

Problem 1: Basics of Haskell (10 Marks)

A. *Working with 'List' in Haskell*: Define a function *m* that takes a lists of lists of integers and sums the numbers in each of the innermost lists together before multiplying the resulting sums with each other. For example,

```
*Main> m [[1,2],[3,5]]
24
*Main> m [[1,2,3],[7]]
42
*Main> m [[7,8],[]]
0
*Main> m [[1,2],[2,3],[4,5,6]]
225
```

B. *Working with 'Basic Functions' in Haskell*: Write a function called greatest, which has the following signature:

```
greatest :: (a -> Int) -> [a] -> a
```

`greatest f seq` returns the item in `seq` that maximizes function `f`. For example:

```
*Main> greatest sum [[2,5], [-1,3,4], [2]]
  [2,5]

*Main> greatest length ["the", "quick", "brown", "fox"]
  "quick"

*Main> greatest id [51,32,3]
  51
```

NB: If more than one item maximizes f, then greatest f returns the *first* one.

C. *Custom List Data Type*: Implement `toList :: [a] -> List a`, which converts a regular Haskell list to a `List a`; and vice versa. For example,

```
*Main> toList []
  Empty

*Main> toList [2, 7, 4]
```

```
  Cons 2 (Cons 7 (Cons 4 Empty))

*Main> toList "apple"
  Cons 'a' (Cons 'p' (Cons 'p' (Cons 'l' (Cons 'e' Empty))))
```

AND

```
*Main> toHaskellList Empty
  []

*Main> toHaskellList (Cons 2 (Cons 7 (Cons 4 Empty)))
  [2,7,4]

*Main> toHaskellList (Cons "cat" (Cons "bat" (Cons "rat" Empty)))
  ["cat","bat","rat"]
```

Problem 3: Anagram (10 Marks)

Once Dylan and Lynda are playing with cards made by themselves only. Each card has some letter written on it and let us assume they have made enough cards so that they will not cause any limit in this game.

Now Dylan and Lynda bring up their own cards and they put all the cards on the table.

Let's say Dylan brought up 3 cards and Lynda brought up only one card.

Dylan has 3 cards namely 'X', 'Y', 'Y 'and Lynda has brought 'X '(order is important)

Together they have 'X', 'Y', 'Y, 'X'.

Now, if you have observed then you would have noticed both of them has special characteristic in their names. Both names can form Anagrams (Dylan and Lynda) hence they always love anagram pairs. Now in these cards they decided to count total anagram subset of cards.

For Eg. here total 4 anagram subsets are possible (X, X), (Y, Y), (XY, XY), (XYY, YYX)

(Remember they have followed order of the cards).

Since they both are very lazy it's your job to help them to find total anagram subsets in the cards.

Example:

```
*Anagram> [['x', 'y'], ['x', 'y']]
  5
Explanation: 6 possible anagram pairs after joining the both
strings are: (x, x), (y, y), (xy, xy), (xy, yx), (yx, xy)

*Anagram> [['x', 'y', 'y'], ['x']]
  4
Explanation: 4 possible anagram pairs after joining the both
strings are: (x, x), (y, y), (xy, yx), (xyy, yyx)
```

Problem 3: IITG Football League (30 Marks)

The IITG sports board has decided to promote a healthy campus lifestyle in the campus amongst the academic personal. So they have decided to inaugurate a yearly football league amongst the various academic departments of IITG. The event will start from 1st Nov to 7th Nov 2019. The sports board has received positive response from the campus residents. 12 teams have registered for the inaugural season out of which 11 teams are from different departments and 1 team is from the staff members. The tournament will be knock out in the first round followed by a league in the second round out of which top 4 teams will reach the playoffs

Write a program in Haskell to carry out the draw for the fixtures. Your program should consider the following

a. The draw will be for the initial round only where each team will play only one match.
b. Use acronyms of the registered teams as follows:

Sl. No	Team	Acronym
1	Biosciences & Bio-Engineering	BS
2	Chemical Engineering	CM
3	Chemistry	CH
4	Civil Engineering	CV
5	Computer Science & Engineering	CS
6	Design	DS
7	Electronics and Electrical Engineering	EE
8	Humanities and Social Sciences	HU
9	Mathematics	MA
10	Mechanical Engineering	ME
11	Physics	PH
12	Staff Members	ST

c. After the draw, generate the fixtures for the first round with date and time.
d. The first round will commence from 1st to 3rd Nov 2019. Schedule the fixtures in this period only.
e. There will be two matches per day. The first match will start at 9:30 AM and the second match will start at 7:30 PM
f. Your program should be able to display the entire fixture, match details about any particular team and the next match details

Sample Input/ Output:

```
*Main> fixture 'all'
  CS vs PH      1-11      9:30
  MA vs ST      1-11      7:30
        ----
        ----
  CV vs DS      3-11      7:30

*Main> fixture 'DS'
  ME vs DS      3-11      7:30

*Main> nextMatch 1  13.25    [This means what is the next match when
                              current date is 1/11 and current time is 1:15 PM]
  MA vs ST      1-11    7:30
```

Problem 4: Puzzle Solving (40+10 Marks)

Assume that there is a 4x4 grid structure as shown in the Fig 1(A). Some of the cells in the grid have some symbols (from a set of 4 distinct symbols), others are blank. The objective of the puzzle is to fill in the blank cells with symbols from the set of the four symbols, in such a way that every row, every column and the four 2x2 corner blocks have exactly one occurrence of each symbol.

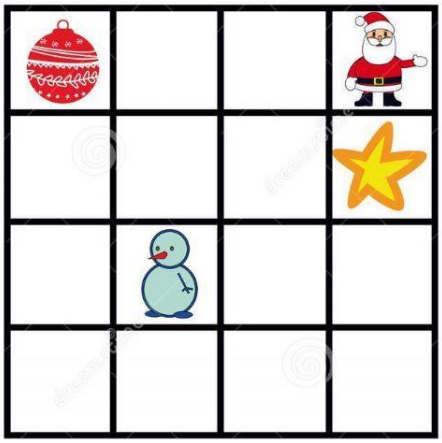

Fig 1(A): Example Input

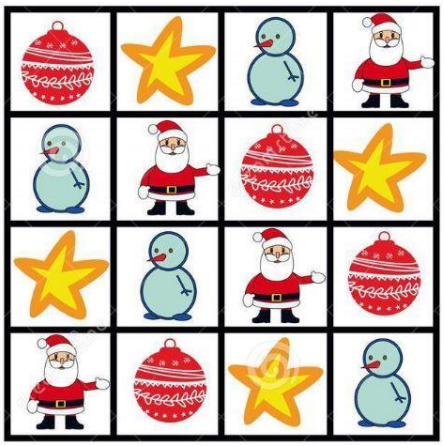

Fig 1B: Example Output

Input: A partially filled up grid structure like Fig 1(A), where each symbol should be used only once. There should be scope of providing user-input (for the partial filling).

Output: Not solvable / the solution like Fig 1(B).

Extension for Bonus Point (30): Solve the same for a 9x9 grid. Like the earlier one, some of the cells in the grid have some symbols, whereas others are kept blank. However, this time the number of distinct symbols are 9 instead of 4. The objective is same as before, i.e., to fill in the blank cells with symbols from the set of the 9 symbols, in such a way that every row, every column and each of the nine 3x3 blocks, shown in Fig 1(C), has exactly one occurrence of each symbol.

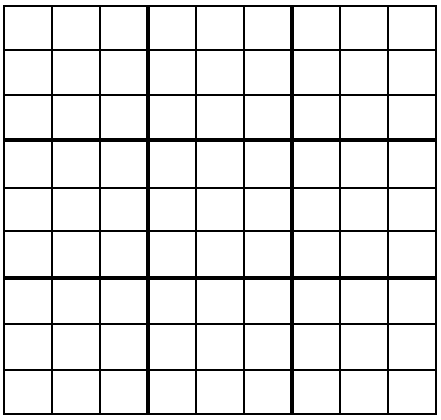

Fig 1(C): 9x9 blocks and the 3x3 sub-blocks

Input: A partially filled up grid structure– there should be scope of providing user-input (for the filling).

Output: Not solvable / the solved puzzle**.**