PGP Coursework 1 for 2021: *Haskell Imp*

Z	.	2
1	#	.
A	.	.

In this assignment you will implement the *Haskell Imp* game in Haskell. You may use any functions in Prelude to do this, but follow the steps below. It is possible to complete this task and get full marks by using the material covered in just Chapters 2 to 6 of the Haskell lecture notes.

- Deadline for submission: **4pm on 20 April 2021**
- Submit via Moodle as a Haskell file which includes all your functions. Ensure your student ID is part of the file name and use the suffix `.hs`.
- All your functions should compile without error when loaded into GCHi (version 8.0 or higher).
- Total of 10 marks.

Rules of Play

- An Imp lives on a rectangular grid and each turn can move north (N), east (E), south (S) or west (W).
- The Imp starts with finite fuel and each turn uses 1 unit of fuel to move.
- The Imp is not able to move through walls, out of the grid or back to a position it has already visited.
- The Imp's goal is to get from the start position to the end position, collecting as many jewels as possible on the way, before running out of fuel.
- If all the fuel is used up before reaching the end, the Imp dies and end of game!

Objective

Write a Haskell program that given a grid (populated with start and end positions, walls and jewels) and fuel, will find the best route for the Haskell Imp.

Grid representation

The grid is represented as a list of character strings where the starting position is represented by the character 'A', the end by 'Z', a wall by '#', blank spaces by '.' and different numbers of jewels by '1', '2' and '3'.

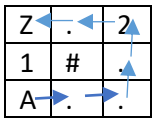
So, e.g.,

```
map1 :: [[Char]]
map1 = ["Z.2",
        "1#.",
        "A.."]
```

represents the 3-by-3 grid,

Z	.	2
1	#	.
A	.	.

with starting point at A, one wall in the middle, 1 jewel in the leftmost column, 2 jewels at the top-right and the end at Z. If initial fuel given is 6, then the best path is to go east, east, north, north, west and west, since 2 jewels are collected:



Paths are represented as strings of direction characters, e.g. `"EENNWW"` in this case.

Coordinate positions are given on the grid as (x,y) with x-axis representing rows, y-axis representing columns and range of values starting from 0. For example, the 'A' in the above grid is at position (2,0).

Step-by-step instructions to implement the Imp game are given below. Marks are allocated for both a correctly operating program and for programming style. Remember to include type definitions for all functions you include in your program.

Hint: The model answer takes 48 lines of code, not including comments.

Instruction steps

To get started, here are two functions to help manipulate the game grids:-

```
posOnMap :: [[Char]] -> (Int,Int) -> Bool
posOnMap vss (x,y) = x>=0 && x<length(vss)
                     && y>=0 && y<length(vss!!x)

getAtPos vss (x,y) | posOnMap vss (x,y) = (vss!!x)!!y
                   | otherwise          = '#'
```

The function `posOnMap` will confirm whether the coordinate position given by tuple `(x,y)` is a legal position on the grid given by `vss`.

The function `getAtPos` will return the value or type of item at position `(x,y)` on grid `vss`. Notice that if `(x,y)` is not a legal position, a wall (`'#'`) is returned.

Complete the following steps. Please note that the steps include examples based on `map1`. However, your functions should also work on any other grid.

Ensure that you name your functions exactly as shown in the steps below, since an automatic marking system will load your code and will be looking for the correct function names. You will lose marks if functions are not found because they are wrongly named.

1. Deduce the correct data type for `getAtPos` (include the type in your Haskell script). **(1)**

2. Write a function `setAtIndex :: [a] -> Int -> a -> [a]` to change the value of a list at a given index position (first argument) to a new value (second argument).
   For example, `setAtIndex [1,2,3] 1 5` returns `[1,5,3]`. **(1)**

3. Write a function `blockAtPos :: [[Char]] -> (Int,Int) -> [[Char]]` that changes the object at a given position on a grid to a wall (`'#'`) and returns the updated grid.
   For example, `blockAtPos map1 (0,1)` evaluates to `["Z#2","1#.","A.."]`. **(1)**
   Hint: Use `setAtIndex`.

4. Write a function `findStart` that returns the (x,y) position of the starting position (`'A'`) on a grid. Include the correct type definition for `findStart`.
   For example, `findStart map1` returns `(2,0)`. **(2)**

5. Write a function `legalPos :: [[Char]] -> (Int,Int) -> Bool` that takes a grid and a position and returns `True` if only if the position is a legal position for the *Imp* on the grid (note that the Imp is allowed to be on a jewel).
   For example,
   - `legalPos map1 (4,1)` returns `False`, since (4,1) is off the map;
   - `legalPos map1 (2,1)` returns `True`, since (2,0) is an empty position (`'.'`) on the map.
   - `legalPos map1 (1,1)` returns `False`, since (1,1) is wall (`'#'`) on the map.

   (1)

The function `movePos` gives a move from one position, given a direction, to a new position, and is defined as follows:-

```
movePos :: (Int,Int) -> Char -> (Int,Int)
movePos (x,y) 'N' = (x-1,y)
movePos (x,y) 'S' = (x+1,y)
movePos (x,y) 'E' = (x,y+1)
movePos (x,y) 'W' = (x,y-1)
```

6. Use list comprehension to write a function `legalMoves` that returns all legal moves from a position. A legal move is a move to a legal position. You can make use of functions `legalPos` and `movePos` to write this function. It should return a list of characters indicating possible directions.
   For example, `legalMoves map1 (findStart map1)` returns `"NE"` since it is only possible to travel north or east from position `A` on `map1`. **(1)**

7. Write a function `valueAtPos:: [[Char]] -> (Int,Int) -> Int` to return the number of jewels at a given position (0 if there are no jewels) as an integer. **(1)**

8. Write a recursive function

```
bestPath :: [[Char]] -> (Int,Int) -> Int -> ([Char], Int)
```

to find the best path on a grid (1st argument), from a current position (2nd argument) and with current fuel left (3rd argument). It needs to return a tuple with the list of directions as the first element and the total number of jewels found as the second.
For example, `bestPath map1 (1,2) 3` returns `("NWW",2)` **(2)**

Z	.	2
1	#	.
A	.	.

You may make use of any functions you have built so far to build this function.

- Hint 1: Read the Rules of Play carefully to decide how to control the moves, fuel and count number of jewel. Remember the best path is the one where the most jewels can be collected.
- Hint 2: You will most likely need an auxiliary function to help write `bestPath`.

Finally, you may test your code using

```
play :: [[Char]] -> Int -> ([Char], Int)
play vss = bestPath vss (findStart vss)
```

to play a game of *Haskell Imp*.
For example, `play map1 6` plays the game on `map1` with 6 units of fuel. It should evaluate to `("EENNWW",2)`.

Appendix

Other maps you can use to test your functions:

```
map2 = ["Z.1..",
        "####.",
        "1.3..",
        ".#.#.",
        "A..2."]

map3 = ["..2..Z",
        ".####.",
        "..3...",
        "1#..#.",
        ".#.##1",
        "A..1.."]
```