German University in Cairo
Department of Computer Science
Dr. Nourhan Ehab

Concepts of Programming Languages, Spring Term 2022
Project 2: Chess Game

Due: 3rd June 2023

1. Project Description

In this project, you are going to implement part of the engine for a chess game in Haskell. Your implementation will include functions that support initializing a chess board, visualizing the chess board, moving a piece according to its movement rules, and suggesting possible legal moves for any piece on the board. The below figure shows the initial configuration of a chess board. Check `https://www.chess.com/terms/chess-pieces` for a reminder of the chess pieces and their movement rules.

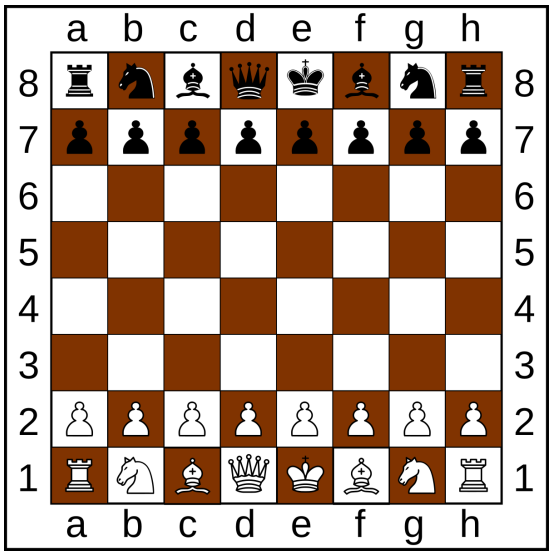

Figure 1: Initial Board Configuration

To keep things simple, you can assume the following:

a) There are no promotions of pawns.

b) It is legal for a king to move to locations in which he will be threatened.

3. Implementation Description. Your implementation must contain the following type definitions.

```
type Location = (Char, Int)
data Player = White | Black deriving (Show, Eq)
data Piece = P Location | N Location | K Location | Q Location | R Location
           | B Location deriving (Show, Eq)
type Board = (Player, [Piece], [Piece])
```

a) `Location` represents a cell on the chess board indexed by a column letter (from a to h) and a row number (from 1 to 8) as shown in Figure 1.

b) `Player` is one of the black or white players.

c) `Piece` is represented by one of the data constructors `P`, `N`, `K`, `Q`, `R`, and `B` which represent a pawn, a knight, a king, a queen, a rook, and a bishop respectively, followed by the piece's location on the board.

d) `Board` is represented by a triple consisting of the player on which the current turn is, a list of the white player's pieces, and a list of the black player's pieces.

Your implementation must also contain the following functions. You are not allowed to change the given type definitions of the functions, but you can add any other helper functions you need.

a) `setBoard :: Board`
The function does not take any inputs and returns a board representing the initial configuration depicted in Figure 1. Assume that the first turn is always on the white player. Example:

```
> setBoard

(White, [R ('h',1),N ('g',1),B ('f',1),K ('e',1),
         Q ('d',1),B ('c',1),N ('b',1),R ('a',1),
         P ('h',2),P ('g',2),P ('f',2),P ('e',2),
         P ('d',2),P ('c',2),P ('b',2),P ('a',2)] ,
         [R ('h',8),N ('g',8),B ('f',8),K ('e',8),
          Q ('d',8),B ('c',8),N ('b',8),R ('a',8),
          P ('h',7),P ('g',7),P ('f',7),P ('e',7),
          P ('d',7),P ('c',7),P ('b',7),P ('a',7)])
```

b) `visualizeBoard:: Board->String` The function takes as input a board and returns a visual representation of the board in a string. Black pieces are suffixed with 'B' and white pieces are suffixed with 'W'. Example:

```
> visualizeBoard (setBoard)
    a    b    c    d    e    f    g    h
8 | RB | NB | BB | QB | KB | BB | NB | RB |
7 | PB | PB | PB | PB | PB | PB | PB | PB |
6 |    |    |    |    |    |    |    |    |
5 |    |    |    |    |    |    |    |    |
4 |    |    |    |    |    |    |    |    |
3 |    |    |    |    |    |    |    |    |
2 | PW | PW | PW | PW | PW | PW | PW | PW |
1 | RW | NW | BW | QW | KW | BW | NW | RW |

Turn: White
```

c) `isLegal:: Piece -> Board -> Location -> Bool`
The function takes as input a piece, a board, and a location. It returns `True` if the move of the piece on the given board to the input location is legal, and `False` otherwise.
Example:

```
> isLegal (P ('a',7)) (setBoard) ('a',5)
True

> isLegal (P ('a',7)) (setBoard) ('a',4)
False

> isLegal (R ('h',8)) (setBoard) ('h',7)
False
```

d) `suggestMove:: Piece -> Board -> [Location]`
The function takes as input a piece and a board and outputs a list of possible legal next locations for the piece.
Example:

```
> suggestMove (P ('e',2)) (setBoard)
[('e',3),('e',4)]

> suggestMove (N ('b',8)) (setBoard)
[('a',6),('c',6)]
```

e) `move:: Piece -> Location -> Board -> Board` The function takes as input a piece, a location, and a board and returns a new updated board after the move is applied if it is a legal move. Otherwise, if the move is illegal, the function throws an appropriate error.

```
> move (P ('a',7)) ('a',6) (setBoard)
Program error: This is White player's turn,  Black can't move.

> move (R ('h',1)) ('h',2) (setBoard)
Program error: Illegal move for piece R ('h',1)

> move (N ('b',3)) ('d',4) (White, [R ('h',1),N ('g',1),B ('f',1),
        K ('e',1), Q ('d',1),B ('c',1),N ('b',3),R ('a',1),
         P ('h',2),P ('g',2),P ('f',2),P ('e',2),
         P ('d',2),P ('c',2),P ('b',2),P ('a',2)] ,
         [R ('h',8),N ('g',8),B ('f',8),K ('e',8),
          Q ('d',8),B ('c',8),N ('b',8),R ('a',8),
          P ('h',7),P ('g',7),P ('f',7),P ('e',7),
          P ('d',7),P ('c',7),P ('b',7),P ('a',7)])

(Black,[R ('h',1),N ('g',1),B ('f',1),K ('e',1),
            Q ('d',1),B ('c',1),N ('d',4),R ('a',1),
            P ('h',2),P ('g',2),P ('f',2),P ('e',2),
            P ('d',2),P ('c',2),P ('b',2),P ('a',2)],
             [R ('h',8),N ('g',8),B ('f',8),K ('e',8),
```

```
Q ('d',8),B ('c',8),N ('b',8),R ('a',8),
P ('h',7),P ('g',7),P ('f',7),P ('e',7),
P ('d',7),P ('c',7),P ('b',7),P ('a',7)])
```

3. **Teams.** You are allowed to work in teams of four members. You must stick to the same team you worked with in Project 1. IDs for the submitted teams are posted on the CMS.

5. **Deliverables.** You should submit a single `.hs` file named with your team ID containing your implementation. The submission link will be posted on the CMS prior to the submission.