# Homework 03 (Due: Friday, October 30, 2020, 11 : 59 : 00PM Central Time)

CSCE 322

## Contents



## List of Figures



## 1 Instructions

In this assignment, you will be required to write Haskell functions that facilitate playing with the Boss Puzzle. You will be provided with skeleton code that includes functionality for reading from a data file and generating outputs. The code will also include function declarations that you must define.

### 1.1 Data File Specification

An example of a properly formatted file is shown in Figure 1. The tuple represents moves to be made and a puzzle.

```
(
"uuurlluddrdruu",
[
[0,1,6],
[7,5,4],
[2,8,3]
]
)
```

Figure 1: A properly formatted Boss puzzle encoding

The tuple contains a list of moves (up, down, left, and right) and a puzzle (where 0 corresponds to the empty space in the puzzle).

### 1.2 csce322hmwrk03prt(num).hs

This assignment requires the implementation of three (3) methods: `oneMove` , `manyMoves` , and `puzzleSolvable` . Each method will be implemented in its own file (named `csce322hmwrk03prt(num).hs`, where `(num)` is 01, 02, or 03). The behavior of each function is described below.

#### 1.2.1 oneMove (csce322hmwrk03prt01.hs)

`oneMove :: Char -> [[Int]] -> [[Int]]` This method will take a move to make and a puzzle, and return a puzzle according to the following rules:

1. If the puzzle is already solved, the move is not made and the puzzle is returned unchanged
2. If the move to make is not valid on the current puzzle, the puzzle is returned unchanged
3. If the puzzle is not solved and the move to make is valid on the current puzzle, the move is made and the updated puzzle is returned

```
(
"uuurlluddrdruu",
[
[0,1,6],
[7,5,4],
[2,8,3]
]
)
```

Figure 2: Puzzle State Before `oneMove`

```
[7,1,6]
[0,5,4]
[2,8,3]
""
```

Figure 3: Puzzle State After `oneMove`

**Sample Sequence**

#### 1.2.2 `manyMoves (csce322hmwrk03prt02.hs)`

`manyMoves :: [Char] -> [[Int]] -> [[Int]]` This method will take a list of moves to make and a puzzle and make each move in accordance to the rules for `oneMove` .

```
(
"urrddlluulrruuudurrrru",
[
[11,13,6,7],
[9,10,14,1],
[0,8,3,4],
[12,5,15,2]
]
)
```

Figure 4: Puzzle State Before `manyMoves`

```
[11,13,6,7]
[10,14,3,1]
[9,8,15,4]
[0,12,5,2]
""
```

Figure 5: Puzzle State After `manyMoves`

**Sample Sequence**

#### 1.2.3 `puzzleSolvable (csce322hmwrk03prt03.hs) (10% Extra Credit)`

`puzzleSolvable :: [[Int]] -> Bool` This method will determine if a puzzle is solvable using the rules laid out in this tutorial .

```
(
"dududrllu",
[
[7,17,16,14,3],
[1,2,13,15,12],
[9,18,11,8,6],
[10,5,0,19,4]
]
)
```

Figure 6: A solvable puzzle

```
(
"luudrdlurlurrdruulluurl",
[
[0,4,3],
[6,5,8],
[1,2,7]
]
)
```

Figure 7: An unsolvable puzzle

**Sample Sequence**

## 2 Naming Conventions

You will be submitting at least 3 `.hs` files (`csce322hmwrk03prt01.hs`, `csce322hmwrk03prt02.hs`, and `csce322hmwrk03prt03.hs`). If you do not submit a modified `Helpers.hs` file, the default one will be provided.

## 3 webgrader Note

Submissions will be tested with `ghc`. `cse.unl.edu` is currently running version 8.6.4 of `ghc`.

## 4 Point Allocation

| Component | Points |
|---|---|
| `csce322hmwrk03prt01.hs` | |
| Compilation | 10 |
| Test Cases | $1 \times 30$ |
| Total | 40 |
| `csce322hmwrk03prt02.hs` | |
| Compilation | 10 |
| Test Cases | $1 \times 50$ |
| Total | 60 |
| Total | 100 |

## 5 External Resources

Learn Haskell Fast and Hard
Learn You a Haskell for Great Good!
Red Bean Software
Functional Programming Fundamentals The Haskell Cheatsheet
Haskell Wikibook
Haskell.org