GRADING ASSIGNMENT (PART 1): SOLITAIRE

There are many variations of the card game "Solitaire", some using a single pack of 52 cards, others using two packs, and more obscure variants using different sets of cards. In this first part of your grading assignment, you are required to set up the framework that will allow later implementation (coming in Part 2...) of particular variants of the game.

If you are unfamiliar with Solitaire in any form, the most common form is known as Klondike, or often just "classic solitaire". It is available as a free game on many platforms and can also be played in a browser from many sites. https://www.classic-solitaire.com/ is one such site, which also does a good job of explaining the rules and strategies.

Other variations include Spider Solitaire (played with two packs of cards), 8-Off Solitaire (a single pack of cards), Canfield Solitaire (also played with a single pack of cards). To get an idea of just how many variants there are, look at the list on Wikipedia https://en.wikipedia.org/wiki/List_of_patience_games (and bear in mind that this is probably *not* comprehensive).

STEP 1: DEFINE YOUR INITIAL DATATYPES

You must define datatypes for:

- A **Suit** (Hearts, Clubs, Spades or Diamonds)
- A **Pip** (Ace, Two, ... King).
- A **Card**
- A **Deck** of Cards

STEP 2: IMPLEMENT SOME BASIC FUNCTIONALITY

First, define Haskell constants and utilities as follows

- pack – a list of all the 52 cards in a standard pack (not including Jokers,)
- sCard – a function which takes a Card and returns the successor card, e.g. the successor of 6❤ is 7❤. Assume that the successor of a King is the Ace in the same suit. For some solitaire games, Ace is low (first card, before Two); for others Ace is high (last card, after King). This function should make no assumptions about it.
- pCard takes a Card and returns its predecessor. Similar to sCard, assume the predecessor of Ace is King in the same suit.
- isAce – a predicate which is True if a given Card is an Ace.
- isKing – a similar predicate for a King.

Next, define a Haskell function **shuffle**, which takes a deck of cards and returns a shuffled deck of same cards. Please refer to lecture material for guidance on this function!

STEP 3: DEFINE DATATYPES TO REPRESENT AN EIGHT-OFF BOARD

Eight-off solitaire is explained in depth in Appendix A to this document. The board captures a snapshot of the cards at any point in the game, including the foundations, columns and reserve. You need to define:

- A **Foundation**
- A **Column**
- A **Reserve**
- A **Board** with a specific type constructor for an **EOBoard**. (Different games of solitaire have different boards; you will create a variant for spider solitaire in step 5.)
- A constant of type Board that represents the eight-off initial layout shown in the screenshot in Appendix A.
- You should make your **Board** datatype an instance of Show (that is, you must write the code to generate the output), such that an **EOBoard** would be displayed like this:

```
EOBoard
Foundations  [(Two,Clubs),(Two,Spades)]
Columns
  [(Seven,Diamonds),(Ace,Hearts),(Queen,Hearts),(King,Clubs),(Four,Spades)]
  [(Five,Diamonds),(Queen,Spades),(Three,Diamonds),(Five,Spades),(Six,Spades),(Seven,Hearts)]
  [(King,Hearts),(Ten,Diamonds),(Seven,Spades),(Queen,Diamonds),(Five,Hearts),(Eight,Diamonds)]
  [(Jack,Spades),(Six,Hearts),(Seven,Clubs),(Eight,Spades),(Ten,Clubs),(Queen,Clubs)]
  [(Eight,Clubs),(Ace,Diamonds),(King,Diamonds),(Jack,Hearts),(Four,Clubs)]
  [(Two,Diamonds),(Three,Hearts),(Three,Clubs),(Ten,Hearts),(Six,Diamonds),(Jack,Clubs)]
  [(Nine,Spades),(Four,Diamonds),(Nine,Clubs),(Nine,Hearts),(Three,Spades),(Ten,Spades)]
  [(Four,Hearts),(Nine,Diamonds),(King,Spades),(Eight,Hearts)]
Reserve      [(Two,Hearts),(Six,Clubs),(Five,Clubs),(Jack,Diamonds)]
```

(This board is the board that would result from the correct application of **toFoundations** (see step 4) to the board given in Appendix A.)

STEP 4: IMPLEMENT FURTHER FUNCTIONALITY

Define Haskell functions as follows:

- A function **eODeal**, which "deals" an opening Board. (That is, it returns the initial board of a game of eight-off solitaire.)
- a function **toFoundations** (sometimes called *autoplay*) which takes an **EOBoard** and returns the **EOBoard** obtained by making all the possible moves to the **Foundations.** In the example given in appendix A, Ace ♣ and Ace ♠ are available to start foundations. Note that this process is recursive: once Ace ♣ has moved, 2♣ will move and that uncovers 2♠.

STEP 5: DEFINE DATATYPES TO REPRESENT A FOUR SUIT SPIDER BOARD

Spider solitaire is explained in depth in Appendix B to this document. Having read about it, consider these questions, and whether your answers will lead you to change the code you have written so far:

1. Spider solitaire uses two decks of cards instead of one. Does this change anything?
2. As for eight-off, a spider solitaire game has columns and foundations. Are they the same as columns and foundations in eight-off solitaire, or do you need to define new types for them?
3. Are there any other changes that you need to make to the basic data types and/or functions that you have already defined in order to support this new variant?
4. Spider solitaire has a **Stock** rather than a reserve. The cards in the stock should not be visible (that is, they remain face down), but it should be possible to see how many more deals are possible from the stock. You can also have face down cards in the columns. How will you manage this?

(You do not have to provide explicit answers to these questions, but you might change your implementation so far based upon the answers to these questions.)

You must define:

- A type for **Stock**
- A type constructor **SBoard** for the **Board** datatype that was defined earlier.
- A constant of type Board that shows the game in progress (that is, the state of the board at that particular move) for the screenshot shown in Appendix B.
- As for **EOBoard**, you should have an instance of Show for an SBoard, such that the board shown in the screenshot in Appendix B would be presented like this:

```
SBoard
Foundations  [(King,Hearts)]
Columns
  [(Eight,Diamonds),(Nine,Hearts)]
  [(Two,Diamonds)]
  [(Ace,Spades),(Two,Spades),(Three,Spades),(Four,Spades),(Five,Spades),(Six,Clubs),
   (Seven,Clubs),(Eight,Clubs),(Nine,Clubs),(Ten,Diamonds),(Jack,Diamonds),(Queen,Diamonds),
   (King,Diamonds),<unknown>,<unknown>]
  [(Seven,Clubs),(Eight,Diamonds),(Nine,Diamonds),(Ten,Diamonds),(Jack,Diamonds),
   (Queen,Diamonds),(King,Diamonds),(Nine,Clubs),(Ten,Hearts),(Jack,Clubs)]
  [(Ace,Hearts),(Two,Hearts),(Three,Hearts),(Four,Hearts),(Five,Hearts),(Six,Diamonds),
   (Seven,Diamonds),(Queen,Clubs),(King,Hearts)]
  [(Two,Diamonds),(Three,Diamonds),(Four,Diamonds)]
  [(Jack,Clubs),(Queen,Clubs),(King,Clubs),(Two,Spades),(Three,Spades),(Four,Diamonds),
   (Five,Diamonds),(Six,Diamonds),(Seven,Hearts),(Eight,Clubs),(Nine,Spades),(Ten,Clubs),
   (Ace,Clubs),(Two,Clubs),(Three,Clubs),(Four,Clubs),(Five,Spades)]
  [(Seven,Spades),(Eight,Spades),(Nine,Spades),(Ten,Spades),(Jack,Spades),(Queen,Spades),
   (King,Spades),<unknown>,<unknown>,<unknown>]
  [(Jack,Hearts),(Queen,Hearts)]
  [(Ace,Clubs),(Two,Clubs)]
Stock  2 Deals remaining
```

Note: You are not required to make the long columns wrap nicely as shown above. For example, the third column above might simply show as:

```
[(Ace,Spades),(Two,Spades),(Three,Spades),(Four,Spades),(Five,Spades),(Six,Clubs),(Seven,Clubs),(Eight
,Clubs),(Nine,Clubs),(Ten,Diamonds),(Jack,Diamonds),(Queen,Diamonds),(King,Diamonds),<unknown>,<unknow
n>]
```

(or wrapping at whatever width your terminal output was set to).

STEP 6: ONE FINAL FUNCTION

The final step in this stage is to implement a function **sDeal**, which "deals" an opening SBoard. (That is, it returns the initial board of a game of spider solitaire.)

ASSESSMENT

This is the first part of your grading assignment. Remember: you will **pass** this module by passing the two threshold quizzes – and I do hope that every one of you will pass. So before you even start your grading assignment, you have 40% for this module. Any marks you get for the grading assignment are added proportionately to this, so if you get 50% for the grading assignment, you will have 70% for the module. You have to work hard to get a first!

This first part of the assignment does not have a heavy weighting but is essential for the later stages. The function **toFoundations** is the most significant part by far and will account for half the marks on functionality for this part.

Element	Weighting
Core functionality (data types and functions)	
Part 1	10%
Part 2	10%
Style – including layout, choice of function and parameter names, logical presentation (incl. ordering) of function definitions, code comments. These marks will be awarded in proportion to the amount of work that you complete. If you only write one function, you will not score highly here no matter how well that one function is written.	20%
Design (to come, in part 2)	10%
Extension work (to come, in part 2)	30%
Written report (1000 words **max**, more concise is encouraged; to come in part 2)	20%

This stage of the grading assignment is not submitted separately; the whole assignment will be submitted by 3pm Thursday 9th December 2021. You are encouraged to seek feedback as you develop your solution, asking the demonstrators or other staff. Bear in mind that there are many students in this module: if you all seek feedback in the week before submission, it will not be possible!

INDIVIDUAL WORK

This assessment is intended to be individual work. Feel free to discuss ideas with your peers but **what you submit should be written by yourself.** You should have completed the unfair means tutorial when you were in first year, but if you need a refresher you can find it at the top of this page in the student handbook (along with further information about unfair means).
https://sites.google.com/sheffield.ac.uk/comughandbook/general-information/assessment/unfair-means

In previous years we have used similar assignments, and we know some students have published their solutions on the web. If you use their code, this is **not your work** and we will take disciplinary action, as set out in the handbook page linked above.

Please do not publish your solutions on the web.

NEED HELP?

The best place for help is in the lab classes – with online support available for those who cannot attend in person.

If you are seeking help, be prepared to explain the design of your functions – for the part, that really only applied to **toFoundations**, but it will be more important in Part 2. Sometimes the help you need will just be understanding what an error message means, or a small misunderstanding, but at other times you will be advised to reconsider the design – sometimes the difficulties arise because you are trying to solve the problem from the wrong angle.

You can also use the discussion board for general questions (but do not post too much detail about your solution – posts are moderated and those with too much detail will be rejected) or you can email Professor Green. If you email Prof. Green, remember:

> email: p.green@sheffield.ac.uk
> subject: COM2108
> body: **No screen shots.** Include text of code (all the code, so that it can be run) and/or error messages rather than screen shots.

As above, you are **encouraged** to seek feedback about your work, particularly in relation to design and style. We will not give out solutions but we want you to have every chance to learn and improve.

APPENDIX A: 8-OFF SOLITAIRE

Eight-off solitaire Is played with a normal deck of playing cards. The layout at the start of a game is below:

The object of the game is to move all the cards to the foundations. The board and rules are as follows:

Foundations (4 piles: complete these piles to win the game)

- Build up each foundation in suit from Ace to King (for example, a 2♥ can be played on an Ace♥).
- You must start a Foundation with an Ace

Tableau (8 columns, initially of 6 cards each)

- Build down in suit sequence (for example, a 10♠ can be played only on a Jack♠).
- The top card of each column (the **head**) is available for play to another column, the foundations or the cells.
- In addition, you can move groups of cards from the head of a column to the head of another column if they are in sequence and if there are enough free cells so that the cards could be moved individually. (e.g. if one column is 6♥, 7♥, Jack♣... another column is 8♥... and there is at least one space left in the reserve cells you can move the 6♥, 7♥, to leave Jack♣... and 6♥, 7♥, 8♥...)
- If you have <8 non-empty columns, and the head of one of these columns is a King or a King-sequence you can move this King or King-sequence in a new column provided you have sufficient space left in the reserve cells.
- You can move a King from the reserve to an empty column.

Cells (or Reserves; 8 cells)

- These are the "cells". These cells are storage locations for cards being played to the foundations and the tableau.
- The game starts with 4 filled cells.
- There is a maximum of 8 cells.
- Cards in the cells can be moved to the foundations and the tableau.
- Cells can only hold one card.

You should play a few games of 8-off to get the idea. You'll find several web sites where you can play online.

Note that we aren't going to do a simulation with graphics, so the order of the foundations, columns and reserve cells is unimportant.

APPENDIX B: SPIDER SOLITAIRE

Spider solitaire is played with *two* decks of playing cards (104 cards). The object of the game is to move all cards to the foundations. The board and rules are as follows:

Foundations (8 piles, complete these to win the game).

- As in eight-off solitaire, each pile has thirteen cards, from Ace to King, of the same suit.
- In spider solitaire, you can only move cards to the foundation when you have a **complete** set for that pile.

Tableau (10 columns)

- Initially, six cards are dealt to the first four piles and five to the remaining ones. Cards are dealt face down, with only the top card in each pile turned face up.
- The top card in each column is available for play to another column, it can be moved onto the successor card of *any* suit. (So 2◆ could be moved onto 3♠ or 3◆.)
- Where there are groups of cards in sequence of the same suit at the top of a column, these groups may be moved as one unit. (So 3◆2◆ could be moved in one go, but 3♠2◆ would require two moves.)
- When a pile is empty, you can fill the space with any card or group of cards.
- When you can make no further useful moves, you must ensure you have no empty columns (even if this means splitting a group), then deal the next 10 cards from the stock.

Stock

- Initially, this is the remaining 50 cards after the tableau has been dealt.
- You can deal the next 10 cards from the stock at any time, provided there are no empty columns.
- When there are no cards remaining in the stock and you can make no useful moves, the game is over.