CPS506 - Assignment

Preamble

For this assignment, you will solve the same problem for **two** of the four languages we studied this semester. One of the languages chosen must be functional (Elixir or Haskell). You are welcome and encouraged to submit assignments in more than two languages! If you do, the best two will be chosen for your assignment marks.

In addition to the general requirements of the assignment, each language comes with its own slightly modified set of language-specific constraints. These specify the format of the input and output, as well as submission instructions for each language. Aside from these, anything you do inside your program is up to you. Use as many helper functions or methods as you want, use any and all syntax you find useful whether we covered it in class or not, with one caveat: you may not use any functionality that is not part of the base installation of each language. No 3rd party libraries.

General assignment description

For this assignment, you will write a program that deals and evaluates two five-card poker hands and chooses a winner. A description of different hands and their ranking can be seen here:

https://www.fgbradleys.com/et_poker.asp

The input to your program is the *first ten values* in a permutation of the integers 1-52 that represents a ***shuffling*** of a standard deck of cards. The order of suits in an unshuffled deck are Clubs, Diamonds, Hearts, Spades. Within each suit, the ranks are ordered from Ace, 2-10, Jack, Queen, King. The table below shows all 52 cards and their respective indices.

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**
Clubs	Ace	2	3	4	5	6	7	8	9	10	Jack	Queen	King
	14	**15**	**16**	**17**	**18**	**19**	**20**	**21**	**22**	**23**	**24**	**25**	**26**
Diamonds	Ace	2	3	4	5	6	7	8	9	10	Jack	Queen	King
	27	**28**	**29**	**30**	**31**	**32**	**33**	**34**	**35**	**36**	**37**	**38**	**39**
Hearts	Ace	2	3	4	5	6	7	8	9	10	Jack	Queen	King
	40	**41**	**42**	**43**	**44**	**45**	**46**	**47**	**48**	**49**	**50**	**51**	**52**
Spades	Ace	2	3	4	5	6	7	8	9	10	Jack	Queen	King

Thus, an input array that started with the integers [38, 48, 11, 6, …] would represent Queen of Hearts, 9 of Spades, Jack of Clubs, 6 of Clubs, and so on.

Your program will accept this permutation array as input and use it to deal two poker hands of 5 cards each in an alternating fashion. I.e., the first card goes to hand 1, the second card

goes to hand 2, the third card goes to hand 1, fourth to hand 2, etc. until each hand has 5 cards. Once dealt, your program will analyze each hand according to the rules from the website above and decide a winner.

Tie Breaking

There will be no ties. If both hands have the same type, we will implement tie breaking based on the ranks of the cards. For example, if both hands are a flush, then the hand with the card of highest rank is declared the winner. If both hands have a pair, then the hand whose pair is higher wins. For example, a pair of Kings beats a pair of Sevens. If both hands have the same pair, i.e. each hand has a pair of threes, then the hand with the next highest card wins. If both hands have the same pair, and the same high card, we check the 2nd highest card. And so on, and so forth.

If, after comparing hands and ranks, it turns out that the hands are identical, we move on to tie-breaking by suit. If both hands have a pair of threes, and all other cards are of the same rank, then the hand with the three of spades will win. Keep in mind that tie breaking based on suit only comes into effect when tie breaking based on rank is impossible. Suits are ranked from lowest to highest in the following order: Clubs < Diamonds < Hearts < Spades.

Language Requirements

For all languages, your program will be driven at the top level by a function/method called **`deal`** that accepts the permutation array described above as input. This function will return the winning hand based on the format described for each language below. Anything else you do inside this **`deal`** function is completely up to you.

1) Smalltalk requirements:

You must create a class called **`Poker`**, with an instance method called **`deal:`** that accepts a simple integer array as input and returns the winning hand. Its usage will be as follows:

```
| deck winner |
deck := Poker new.
winner := deck deal: #(1 2 3 4 5 6 7 8 9 10). “Sample shuffling”
Transcript show: winner; cr.
```

The winning hand will be returned as a ***sorted*** array of strings where each element is a concatenation of the rank and the suit. The suit must be capitalized. Face cards are represented numerically. 11=Jack, 12=Queen, 13=King. An Ace is represented numerically as 1. The winning hand from the above example would be returned as follows:

```
#(‘1C’ ‘3C’ ‘5C’ ‘7C’ ‘9C’).
```

2) Elixir requirements

In a single Elixir file called Poker.ex, define a module called **`Poker`**, with a function called **`deal`** that accepts a list of integers as an argument and returns the winning hand. Its usage will be as follows:

```
winner = Poker.deal [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

The winning hand will be returned as a ***sorted*** array of strings where each element is a concatenation of the rank and the suit. The suit must be capitalized. Face cards are represented numerically. 11=Jack, 12=Queen, 13=King. An Ace is represented numerically as 1. The winning hand from the above example would be returned as follows:

```
["1C", "3C", "5C", "7C", "9C"]
```

Your Poker.ex file must correctly compile using "elixirc Poker.ex" at the command line. When your submission is tested, we will call your deal function from a separate test script. Please ensure your Poker module, and any additional modules you create, compile correctly.

3) Haskell requirements

In a single Haskell file called Poker.hs, define a module called **`Poker`**, with a function called **`deal`** that accepts a list of integers as an argument and returns the winning hand. Its usage will be as follows:

```
winner = Poker.deal [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

The winning hand will be returned as a ***sorted*** array of strings where each element is a concatenation of the rank and the suit. The suit must be capitalized. Face cards are represented numerically. 11=Jack, 12=Queen, 13=King. An Ace is represented numerically as 1. The winning hand from the above example would be returned as follows:

```
["1C", "3C", "5C", "7C", "9C"]
```

Your Poker.hs file must load cleanly into GHCi. When your submission is tested, we will call your deal function from a separate test function. Please ensure your Poker module, and any additional modules you create, compile correctly.

Submission

You may work in teams of **two (2)** for the assignment, with one caveat: you may not work with the same partner more than once. If two people work together for the Smalltalk assignment, those same two people may not work together for any of the other languages.

Smalltalk submission: Submission details for Smalltalk will be announced at a later time. Stay tuned for details.

Elixir submission: Submit your Poker.ex file on D2L. If you're working with a partner, only one partner submits. Clearly indicate, in both your submission and your Poker.ex file (as a comment), who the two team members are.

Haskell submission: Submit your Poker.hs file on D2L. If you're working with a partner, only one partner submits. Clearly indicate, in both your submission and your Poker.hs file (as a comment), who the two team members are.