Карты в руки

Язык: F#

Максимальное количество баллов: 13

Напишите 11 функций (не считая локальных вспомогательных): четыре про подстановки имён и семь про карточный пасьянс. Все типы данных, которые будут нужны, уже объявлены в файле hw2_sample.fs. В этой работе уже можно начинать наслаждаться автоматическим выводом типов. Если тип вашей функции не совпал с тем, что указан в конце этого документа, но является более общим - всё в порядке.

В ваших решениях используйте pattern matching. Методы Value, IsNone, IsEmpty и т. д., обращение к полям записей через ".", а также коллекции, изменяемые данные и другие конструкции, о которых не шла речь в третьей лекции, использовать нельзя. (Запрет не касается выражений if … then … else в целом, то есть не надо все такие выражения заменять на сопоставление по шаблонам). Порядок элементов в списках не имеет значения, если об этом не сказано отдельно.

Эталонное решение, не считая бонусной задачи, содержит примерно 110 строк кода.

1. В этой задаче требуется найти имена, альтернативные официальному, с учётом уменьшительно-ласкательных и прочих форм имён (Например, Мария Петрова может быть и Машей Петровой, и Машенькой Петровой, и даже Машкой Петровой).
    a. (+1) Напишите функцию **`all_except_option`**, которая принимает кортеж **`(string * string list)`**. Если строка не содержится в списке, верните **`None`**. В противном случае, верните **`Some lst`**, где **`lst`** - список, идентичный исходному, но не содержащий исходную строку. Считайте, что исходная строка встречается в списке не более одного раза.
    b. (+1) Напишите функцию **`get_substitutions1`**, которая принимает кортеж из **`string list list`** (substitutions) и **`string`** (s), и возвращает **`string list`**. Результирующий список должен содержать все строки, которые содержатся в таких списках из substitutions, где также содержится s, но не саму строку s. Например:

```
get_substitutions1([["Fred";     "Fredrick"];
["Elizabeth";"Betty"];
["Freddie";"Fred";"F"]], "Fred")
(* answer: ["Fredrick"; "Freddie"; "F"] *)
```

Считайте, что списки из substitutions не содержат повторений. Используйте решение части A и оператор склеивания списков.

c. (+1) Напишите функцию **get_substitutions2**, которая работает аналогично **get_substitutions1**, но использует хвостовую вспомогательную функцию.

d. (+1) Напишите функцию **similar_names**, которая принимает кортеж из **string list list** (substitutions) и полного имени типа **FullName = {first: string; middle: string; last: string}**, и возвращает список полных имён **FullName list**. Результатом будет список таких полных имён, которые могут быть получены из исходного заменой имени (И только имени) на имя из списков substitutions. Результат должен начинаться с исходного полного имени. Пример:

```
similar_names([["Fred"; "Fredrick"];
["Elizabeth";"Betty"];
["Freddie"; "Fred"; "F"]],
{first="Fred"; middle="W"; last="Smith"})
(*   answer:   [{first="Fred";   last="Smith";
middle="W"};
{first="Fredrick"; last="Smith"; middle="W"};
{first="Freddie"; last="Smith"; middle="W"};
{first="F"; last="Smith"; middle="W"}] *)
```

Не удаляйте дубликаты из решения.

2. Напишите программу, которая обеспечивает процесс игры в пасьянс. В начале игры есть колода карт (card-list) и число (goal). В процессе игры карты можно брать в руку (held-cards). За ход игрок может либо вытянуть карту из колоды (удалить первую карту из card-list и добавить в held-cards), либо выбрать любую карту из руки и сбросить её. Игра заканчивается либо когда игрок решает больше не делать ходов, либо когда сумма значений карт в руке превысит

число goal.

Цель - закончить игру с наименьшим счётом (идеальный результат - 0). Счёт изменяется следующим образом: пусть sum - сумма значений всех карт в руке. Если sum > goal, счёт равен 3*(sum - goal), иначе счёт равен (goal - sum). Если все карты в руке одного цвета, то итоговый счёт делится пополам (целочисленно).

a. (+1) Напишите функцию **`card_color`**, которая принимает на вход значение типа **`card`** и возвращает соответствующий цвет (значение типа **`color`**).
b. (+1) Напишите функцию **`card_value`**, которая принимает на вход значение типа **`card`** и возвращает её стоимость: для пронумерованных карт (девятка, семёрка и т. д.) стоимость равна номеру, стоимость туза - 11, стоимость всех остальных карт - 10.
c. (+1) Напишите функцию **`remove_card`**, которая принимает кортеж из списка карт **`cs`**, карты **`c`** и исключения **`e`**, и возвращает список, состоящий из всех элементов **`cs`**, кроме **`c`**. Если в исходном списке содержится несколько значений c, удалите только первое. Если c вообще нет в списке, выбросьте исключение **`e`**.
d. (+1) Напишите функцию **`all_same_color`**, которая принимает список карт и возвращает **`true`**, если все карты списка одного цвета.
e. (+1) Напишите функцию **`sum_cards`**, которая принимает список карт и возвращает их суммарную стоимость. Используйте локально определённую вспомогательную хвостовую рекурсию.
f. (+1) Напишите функцию **`score`**, которая принимает кортеж из списка карт (**`held_cards`**) и числа (**`goal`**) и возвращает счёт, как описано выше.
g. (+3) Напишите функцию **`officiate`**, реализующую описанную выше игру. Она принимает кортеж из списка карт (**`card list`**), списка ходов (**`move list`**), которые сделал игрок, и целого числа goal, и возвращает счёт, который останется после того, как были сделаны ходы. Используйте

определённую локально вспомогательную рекурсивную функцию, которая принимает аргументы, описывающие текущее состояние игры.

- Игра начинается с пустым списком **held_cards**
- Игра заканчивается, если ходов больше не осталось (пустой **move list**)
- Если игрок сбрасывает карту **c**, игра продолжается со списком **held_cards**, без карты **c**. Список карт в колоде не изменяется. Если с не содержится в списке **held_cards**, выбросьте исключение **IllegalMove**.
- Если игрок хочет взять карту из колоды (**draw**), а колода (**card_list**) уже пуста, игра заканчивается. Если после вытаскивания карты из колоды сумма карт на руке превосходит число **goal**, игра тоже заканчивается. В остальных случаях игра продолжается с картой, перенесённой из колоды в руку.

В результате, в файле с решением должны быть следующие функции:

```
val all_except_option :
    string * string list -> string list option
val get_substitutions1 :
    string list list * string -> string list
val get_substitutions2 :
    string list * string -> string list
  type FullName =
    {first: string;
     middle: string;
     last: string;}
val similar_names :string list list * FullName -> FullName
list
  type suit =
    | Clubs
    | Diamonds
    | Hearts
    | Spades
  type rank =
    | Jack
```

```
    | Queen
    | King
    | Ace
    | Num of int
  type card = suit * rank
  type color =
    | Red
    | Black
  type move =
    | Discard of card
    | Draw
  exception IllegalMove
val card_color : card -> color
val card_value : card -> int
val remove_card :
    cs:card list * card * e:System.Exception -> card list
val all_same_color : card list -> bool
val sum_cards : card list -> int
val score : card list * int -> int
val officiate : card list * move list * int -> int
```