НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ

Кафедра інформатики та програмної інженерії

Звіт до лабораторної роботи №3

з курсу

“Мультипарадигменне програмування”

студента 2 курсу
групи ІП-01
Коваленка Микити Артемовича

Викладач:
ас. Очеретяний О. К.

Київ – 2022

1. **Завдання лабораторної роботи**

Завдання 1:

Це завдання пов'язане з використанням “заміни імені”, щоб придумати альтернативні імена. Наприклад, Фредерік Вільям Сміт також може бути Фредом Вільямом Смітом або Фредді Вільямом Смітом. Тільки частина (d) присвячена цьому, але інші проблеми є корисними.

(a) Напишіть функцію all_except_option, яка приймає string і string list. Поверніть NONE, якщо рядка немає у списку, інакше поверніть SOME lst, де lst ідентичний списку аргументів, за винятком того, що рядка в ньому немає. Ви можете вважати, що рядок є в списку щонайбільше один раз. Використовуйте рядок, наданий вам, для порівняння рядків. Приклад рішення становить близько 8 строк.

(b) Напишіть функцію get_substitutions1, яка приймає string list list (список списків рядків, замін) і string s і повертає string list. Результат містить всі рядки, які є в якомусь із списків замін, які також мають s, але сам s не повинен бути в результаті.

приклад: get_substitutions1([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]], «Fred»)

відповідь: ["Fredrick","Freddie","F"]

Припустимо, що кожен список із замінами не має повторів. Результат повторюватиметься, якщо s та інший рядок є в більш ніж одному списку підстановок. приклад:

get_substitutions1([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]], "Jeff")

(* відповідь: ["Jeffrey","Geoff","Jeffrey"] *)

Використовуйте підзадачу (a) і додавання до списку ML (@), але ніяких інших допоміжних функцій. Зразок рішення становить близько 6 рядків.

(c) Напишіть функцію get_substitutions2, схожу на get_substitutions1, за винятком того, що вона використовує хвостову рекурсивну локальну допоміжну функцію.

(d) Напишіть функцію `similar_names`, яка приймає `string list list` із підстановками (як у частинах (b) і (c)) і *повне ім'я* типу {first:string,middle:string,last:string} і повертає список повних імен (тип {first:string,middle:string,last:string} list). Результатом є всі *повні імена*, які ви можете створити, замінивши ім’я (і лише ім’я), використовуючи заміни та частини (b) або (c). Відповідь має починатися з оригінальної назви (тоді мати 0 або більше інших імен).

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

відповідь:

[{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 видаляйте дублікати з відповіді. Підказка: використовуйте локальну допоміжну функцію. Зразок рішення становить близько 10 рядків.

Завдання 2:

У цій задачі йдеться про карткову гру-пасьянс, придуману саме для цього питання. Ви напишете програму, яка відстежує хід гри. Ви можете виконати частини (a)–(e), перш ніж зрозуміти гру, якщо хочете. Гра проводиться з *колодою карт* і ціллю. У гравця є *список карт в руці*, спочатку порожній. Гравець робить хід, витягуючи карту з *колоди*, що означає вилучення першої карти зі *списку карт колоди* і додавання її до списку *карт в руці*, або скидання, що означає вибір однієї з *карт в руці* для видалення. Гра закінчується або тоді, коли гравець вирішує більше не робити ходів, або коли сума значень утриманих карт перевищує ціль.

Ціль – закінчити гру з низьким результатом (0 найкращий результат). Підрахунок балів працює наступним чином: Нехай sum — це сума значень карт, що в руці. Якщо sum більша за goal, *попередній рахунок* = 3*(sum − goal), інакше *попередній рахунок* = (goal − sum). Кінцевий рахунок дорівнює *попередньому рахунку*, якщо всі картки, які на руці, не однакового кольору. Якщо всі картки одного кольору, кінцевий рахунок є попереднім рахунком, поділеним на 2 (і округлений, за допомогою цілочисельного ділення; використовуйте оператор div ML)

(a) Напишіть функцію card_color, яка бере карту і повертає її колір (піки і трефи чорні, бубни і чирви червоні). Примітка: достатньо одного case-виразу.

(b) Напишіть функцію card_value, яка бере карту та повертає її значення (нумеровані карти мають свій номер як значення, тузи — 11, все інше — 10). Примітка: достатньо одного case-виразу.

(c) Напишіть функцію remove_card, яка бере список карт *cs*, картку *c* та виняток *e*. Функція повертає список, який містить усі елементи *cs*, крім *c*. Якщо *c* є у списку більше одного разу, видаліть лише перший. Якщо *c* немає у списку, поверніть виняток *e*. Ви можете порівнювати карти з =.

(d) Напишіть функцію all_same_color, яка приймає список карт і повертає true, якщо всі карти в списку мають однаковий колір.

(e) Напишіть функцію sum_cards, яка бере список карт і повертає суму їх значень. Використовуйте локально визначену допоміжну функцію, яка є хвостово-рекурсивною.

(f) Напишіть функцію score, яка отримує на вхід card list (картки, що утримуються) та int (ціль) і обчислює рахунок, як описано вище.

(g) Напишіть функцію officiate, яка «запускає гру». Вона приймає на вхід card list (список карт), move list (що гравець «робить» у кожній точці) та int (ціль) і повертає рахунок у кінці гри після обробки (частину чи всі) переміщення в списку переміщень по порядку. Використовуйте локально визначену рекурсивну допоміжну функцію, яка приймає кілька аргументів, які разом представляють поточний стан гри. Як описано вище:

• Гра починається з того, що утримувані карти є порожнім списком.

• Гра закінчується, якщо більше немає ходів. (Гравець вирішив зупинитися, оскільки move list порожній.)

• Якщо гравець скидає якусь карту c, гра продовжується (тобто виконується рекурсивний виклик), коли утримувані карти не мають c, а список карт залишається незмінним. Якщо c немає в картках, що утримуються, поверніть виняток IllegalMove.

• Якщо гравець бере, але список карт (уже) порожній, гра закінчена. Інакше, якщо розіграш призведе до того, що сума карт, що тримаються, перевищує ціль, гра закінчується (після розіграшу). В іншому випадку гра продовжується з більшою кількістю карт на руці та меншою колодою.

Типове рішення для (g) містить менше 20 рядків.

2. Виконання

Вихідний код:

```
(* if you use this function to compare two strings (returns true if the same
   string), then you avoid several of the functions in problem 1 having
   polymorphic types that may be confusing *)
fun same_string(s1 : string, s2 : string) =
    s1 = s2;


(* put your solutions for problem 1 here *)


fun all_except_option(str : string, lst : string list) =
   let
      fun tail_traverse([]) = []
        | tail_traverse(x::xs) =
```

```
            if same_string(str, x) then tail_traverse (xs) else ([x] @ tail_traverse
(xs))
      val result = tail_traverse(lst)
   in
      if result <> lst then SOME result else NONE
   end

fun get_substitutions1([], _) = []
  | get_substitutions1((s::xs), str : string) =
         (case all_except_option (str, s) of
            SOME sublist => sublist
          | NONE =>  []) @ get_substitutions1 (xs, str);

fun get_substitutions2(lst, str : string) =
   let
      fun tail_traverse([]) = []
        | tail_traverse((s::xs)) =
         (case all_except_option (str, s) of
            SOME sublist => sublist
          | NONE =>  []) @ tail_traverse (xs);
   in
      tail_traverse (lst)
   end;

fun similar_names(lst, { first : string, middle : string, last : string }) =
   let
      fun tail_traverse([]) = []
        | tail_traverse(x::xs) =
            [{ first = x, middle = middle, last = last }] @ tail_traverse (xs)
   in
```

```
      [{first = first, middle = middle, last = last}] @ tail_traverse
(get_substitutions1 (lst, first))
   end;

(* you may assume that Num is always used with values 2, 3, ..., 10
   though it will not really come up *)
datatype suit = Clubs | Diamonds | Hearts | Spades;
datatype rank = Jack | Queen | King | Ace | Num of int;
type card = suit * rank;

datatype color = Red | Black;
datatype move = Discard of card | Draw;

exception IllegalMove;

(* put your solutions for problem 2 here *)
fun card_color(suit, _) =
   case suit of
      (Spades | Diamonds) => Black
    | (Hearts | Clubs) => Red;

fun card_value(_, rank) : int =
   case rank of
      Ace => 11
    | Num n => n
    | _ => 10

fun remove_card(cards : card list, c : card, e) =
   let
      fun tail_traverse([]) = []
        | tail_traverse(x::xs) =
```

```
            if c = x then xs else ([x] @ tail_traverse (xs))
      val result = tail_traverse (cards)
   in
      if result <> cards then result else raise e
   end

fun all_same_color([]) = true
  | all_same_color([x]) = true
  | all_same_color(x::y::xs) = (card_color (x) = card_color (y)) andalso
all_same_color ([y] @ xs)

fun sum_cards(lst : card list) =
   let
      fun tail_traverse([]) = 0
        | tail_traverse(x::xs) =
            card_value (x) + tail_traverse (xs)
      val result = tail_traverse (lst)
   in
      result
   end

fun score(lst : card list, goal : int) =
   let
      val cards_sum = sum_cards (lst)
      val preliminaryScore = if cards_sum <= goal then (goal - cards_sum) else (3 *
(cards_sum - goal))
   in
      if all_same_color(lst) then (preliminaryScore div 2) else preliminaryScore
   end

fun officiate(cards : card list, moves : move list, goal : int) =
```

```
   let
      fun process_move(cards : card list, hand : card list, moves : move list) =
         case moves of
            [] => score (hand, goal)
          | (m::ms) => case m of
               Draw => (case cards of
                  [] => score (hand, goal)
                | (c::cs) => if sum_cards ([c] @ hand) > goal
                             then score([c] @ hand, goal)
                             else process_move (cs, [c] @ hand, ms))
             | Discard card => case hand of
                  [] => raise IllegalMove
                | _ => process_move (cards, remove_card (hand, card, IllegalMove), ms)
   in
      process_move (cards, [], moves)
   end;


use "./functions.sml";


fun assert (expected, actual) =
    if expected = actual then
        true
    else
        raise Fail "assert failure.";


(*1*)
assert (
    [
        {first="Fred", last="Smith", middle="W"},
        {first="Fredrick", last="Smith", middle="W"},
```

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

(*2*)
assert (8, officiate(
    [
        (Spades, Jack),
        (Spades, Ace),
        (Clubs, Ace),
        (Hearts, Num(2))
    ],
    [
        Draw,
        Discard(Spades, Jack),
        Draw,
        Draw,
        Draw,
        Discard (Hearts, Num(2))
    ], 30));

assert (6, officiate(
    [
        (Spades, Jack),
        (Spades, Ace),
```

```
            (Clubs, Ace),
            (Hearts, Num(2))
        ],
        [
            Draw,
            Discard(Spades, Jack),
            Draw,
            Draw,
            Draw,
            Discard (Hearts, Num(2))
        ], 20));

assert (3, officiate(
        [
            (Spades, Jack),
            (Spades, Ace),
            (Diamonds, Ace),
            (Diamonds, Num(2))
        ],
        [
            Draw,
            Discard(Spades, Jack),
            Draw,
            Draw,
            Draw,
            Discard(Hearts, Num(2))
        ], 20))
```