Міністерство освіти і науки України
Національний технічний університет України «КПІ ім. Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Мультипарадигменне програмування

ЗВІТ

до лабораторних робіт

Виконав студент ІП-01 Ніколаєв Іван Романович _____
(№ групи, прізвище, ім'я, по батькові)

Прийняв ас. Очеретяний О. К. _____
(посада, прізвище, ім'я, по батькові)

Київ 2021

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.

Програмний код

(task.sml)

```
(* 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 *)
(* a *)
fun all_except_option(str, strlist) =
    let fun recurse(strlist, resList, isFound) =
        case strlist of
```

```
            [] => (resList, isFound)
            |(hd::tl) => if (same_string(hd, str)) then
                            recurse(tl, resList, true)
                        else
                            recurse(tl, hd::resList, isFound)
    in
        let fun rev_list(lst, resList) =
            case lst of
                [] => resList
                | hd::tl => rev_list(tl, hd::resList)
        in
            case recurse(strlist, [], false) of
                (hd::tl, true) => SOME(rev_list(hd::tl, []))
                |(list, false) => NONE
                |([], true) => SOME([])
        end
    end
;

fun get_substitutions1(list, s) =
  case list of
         [] => []
         | x :: x' => case all_except_option(s, x)
                      of SOME findList => findList @ get_substitutions1(x', s)
                      | NONE => get_substitutions1(x', s)
;
fun get_substitutions2(list, s) =
   let fun recurse(currList, find) =
       case currList of
         [] => find
         | x :: x' => case all_except_option(s, x) of
                NONE => recurse(x', find)
              | SOME findList => recurse(x', find @ findList )
   in
      recurse(list, [])
   end
;
fun similar_names(list: string list list, {first = firstname, middle =
middlename, last = lastname}) =
   let fun find_comb(list, comb) =
      case list of
          [] => comb
          | x :: x' => find_comb(x',  comb @ ({first = x, middle = middlename,
last = lastname} :: []))
   in
      {first = firstname, middle = middlename, last = lastname} ::
find_comb(get_substitutions1(list, firstname), [])
   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 *)

(* 2.a *)
fun card_color(card) =
   case card of (Diamonds, _) => Red
              |(Hearts, _) => Red
              | _ => Black
;
(* 2.b *)

fun card_value(card) =
   case card of (_, Ace) => 11
              | (_, King) => 10
              | (_, Queen) => 10
              | (_, Jack) => 10
              | (_, Num x) => x
;
(* 2.c *)

fun remove_card(cs, c, e) =
   let fun recurse(cs, acc, isFound) =
      case cs of
            [] => (acc, isFound)
            |(x::xs) => if x = c then (acc @ xs, true) else recurse(xs, x::acc,
isFound)
    in
     case recurse(cs, [], false) of
            (_, false) => raise e
            | ([], true) => []
            | (x::xs, true) => x::xs
    end
;
(* 2.d *)

fun all_same_color(cards) =
   let val color = case cards of
                        [] => Red
                        | x :: x' => card_color(x)
   fun recurse(cards, color) =
      case cards of [] => true
                  | x :: x' => if card_color(x) <> color then
                                    false
```

```
                                else
                                  recurse(x', color)
   in
      recurse(cards, color)
   end
;
fun sum_cards(cards) =
   let fun sum(cards, total) =
      case cards of
          [] => total
          | x :: x' => sum(x', card_value(x) + total)
   in
      sum(cards, 0)
   end
;
fun score(cards, goal) =
   let val sum = sum_cards(cards)
       val pre_score = if sum > goal then
                        3 * (sum - goal)
                       else
                        goal - sum
   in
      if not (all_same_color(cards)) then
         pre_score
      else
         pre_score div 2
   end
;
fun officiate(cards, moves, goal) =
   let fun next_move(playerCards, moves, allCards) =
       if sum_cards playerCards > goal
       then score (playerCards, goal)
       else
      case moves of
          [] => score (playerCards, goal)
        | x :: x' => case x of
              Discard c => next_move(remove_card (playerCards, c, IllegalMove),
x', allCards)
            | Draw => case allCards of
                    [] => score (playerCards, goal)
                  | j :: j' => next_move(j :: playerCards, x', j')
   in
      next_move([], moves, cards)
   end
;
```

Тести

(test.sml)

```
use "task.sml"
;

fun test(function_name : string, true_result, fact_result) =
    if true_result = fact_result
    then (function_name, "Ok")
    else (function_name, "Failed");
;


(* 1 *)
(* a *)
test("all_except_option", SOME ["1", "3", "4"], all_except_option("2", ["2", "1",
"3", "4"]));
test("all_except_option", SOME [], all_except_option("2", ["2"]));
test("all_except_option", NONE, all_except_option("2", ["1", "3", "4"]));


(* b *)
test("get_substitutions1", ["Fredrick","Freddie","F"],
get_substitutions1([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","
F"]], "Fred"));
test("get_substitutions1", ["Jeffrey","Geoff","Jeffrey"],
get_substitutions1([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffr
ey"]], "Jeff"));
test("get_substitutions1", [],
get_substitutions1([["Fred","Fredrick"],["Jeff","Jeffrey"]], "Vlad"));

(* c *)
test("get_substitutions2", ["Fredrick","Freddie","F"],
get_substitutions2([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","
F"]], "Fred"));
test("get_substitutions2", ["Jeffrey","Geoff","Jeffrey"],
get_substitutions2([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffr
ey"]], "Jeff"));
test("get_substitutions2", [],
get_substitutions2([["Fred","Fredrick"],["Jeff","Jeffrey"]], "Vlad"));


(* d *)
test("similar_names",
    [{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"}));

test("similar_names",
    [{first="Fred", last="Smith", middle="W"}],
```

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

test("similar_names",
    [{first="Fred", last="Smith", middle="W"}],
    similar_names([["Elizabeth","Betty"]], {first="Fred", middle="W",
last="Smith"}));



(* 2 *)
val test_card1 = (Hearts, Jack);
val test_card2 = (Clubs, Num 8);
val test_card3 = (Diamonds, Ace);

val test_card_list1 = [test_card1, test_card2, test_card3];
val test_card_list2 = [test_card1, test_card3];
val test_card_list3 = [];


(* a *)
test("card_color", Red, card_color(test_card1));
test("card_color", Black, card_color(test_card2));
test("card_color", Red, card_color(test_card3));

(* b *)
test("card_value", 10, card_value(test_card1));
test("card_value", 8, card_value(test_card2));
test("card_value", 11, card_value(test_card3));


(* c *)
test("remove_card", [test_card2, test_card3], remove_card(test_card_list1,
test_card1, IllegalMove));
test("remove_card", [], remove_card(test_card_list2, test_card2, IllegalMove));
test("remove_card", [], remove_card(test_card_list3, test_card2, IllegalMove));


(* d *)
test("all_same_color", false, all_same_color(test_card_list1));
test("all_same_color", true, all_same_color(test_card_list2));
test("all_same_color", true, all_same_color(test_card_list3));


(* e *)
test("sum_cards", 29, sum_cards(test_card_list1));
test("sum_cards", 21, sum_cards(test_card_list2));
test("sum_cards", 0, sum_cards(test_card_list3));


(* f *)
test("score", 3, score(test_card_list1, 28));
```

```
test("score", 1, score(test_card_list2, 23));
test("score", 2, score(test_card_list1, 31));

(* g *)
test("officiate", 3, officiate(test_card_list1, [Draw, Draw, Draw, Draw], 28));
test("officiate", 5, officiate(test_card_list1, [Draw, Draw, Discard(Clubs, Num
8)], 20));
test("officiate", 9, officiate(test_card_list1, [Draw, Draw], 15));
```

```
val it = ("all_except_option","Ok") : string * string

val it = ("all_except_option","Ok") : string * string

val it = ("all_except_option","Ok") : string * string

val it = ("get_substitutions1","Ok") : string * string

val it = ("get_substitutions1","Ok") : string * string

val it = ("get_substitutions1","Ok") : string * string

val it = ("get_substitutions2","Ok") : string * string

val it = ("get_substitutions2","Ok") : string * string

val it = ("get_substitutions2","Ok") : string * string

val it = ("similar_names","Ok") : string * string

val it = ("similar_names","Ok") : string * string

val it = ("similar_names","Ok") : string * string
```

```
val it = ("card_color","Ok") : string * string

val it = ("card_color","Ok") : string * string

val it = ("card_color","Ok") : string * string

val it = ("card_value","Ok") : string * string

val it = ("card_value","Ok") : string * string

val it = ("card_value","Ok") : string * string

val it = ("remove_card","Ok") : string * string
```

```
val it = ("all_same_color","Ok") : string * string

val it = ("all_same_color","Ok") : string * string

val it = ("all_same_color","Ok") : string * string

val it = ("sum_cards","Ok") : string * string

val it = ("sum_cards","Ok") : string * string

val it = ("sum_cards","Ok") : string * string

val it = ("score","Ok") : string * string

val it = ("score","Ok") : string * string

val it = ("score","Ok") : string * string

val it = ("officiate","Ok") : string * string

val it = ("officiate","Ok") : string * string

val it = ("officiate","Ok") : string * string
```