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 Шпилька Владислав Сергійович |  |  |
|  |  | (№ групи, прізвище, ім’я, по батькові ) |  |  |
|  |  |  |  |  |
| **Прийняв** |  | ас. Очеретяний О. К. |  |  |
|  |  | (посада, прізвище, ім’я, по батькові ) |  |  |

Київ 2021

**Завдання:**

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.

**Реалізація:**

(\* 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

(\* a \*)

fun all\_except\_option(str, strlist) =

    let fun help\_fun(strlist, acc, isFound) =

        case strlist of

            [] => (acc, isFound)

            |(x::xs) => if (same\_string(x, str)) then help\_fun(xs, acc, true) else help\_fun(xs, x::acc, isFound)

    in

        let fun rev(lst, acc) =

            case lst of

            [] => acc

            | x::xs => rev(xs, x::acc)

        in

            case help\_fun(strlist, [], false) of

            (\_, false) => NONE

            | ([], true) => SOME([])

            | (x::xs, true) => SOME(rev(x::xs, []))

        end

    end

(\* b \*)

fun get\_substitutions1(strlists, str) =

    case strlists of

    [] => []

    |(x::xs) =>

        case all\_except\_option(str, x) of

        SOME (y::ys) =>  (y::ys) @ get\_substitutions1(xs, str)

        | \_ =>  get\_substitutions1(xs, str)

(\* c \*)

fun get\_substitutions2(strlists, str) =

    let fun help\_fun(strlists, acc) =

        case strlists of

        [] => acc

        |(x::xs) =>

            case all\_except\_option(str, x) of

            SOME (y::ys) =>  help\_fun(xs, acc @ (y::ys))

            | \_ =>  help\_fun(xs, acc)

    in

        help\_fun(strlists, [])

    end

(\* d \*)

fun similar\_names(strlists, fullname) =

    let fun help\_fun(simnames) =

        case simnames of

        [] => []

        |(l::ls) =>

            case fullname of

           {first=x,middle=y,last=z} => {first=l, middle=y, last=z} :: help\_fun(ls)

    in

        case fullname of

            {first=x,middle=y,last=z} => fullname::help\_fun(get\_substitutions2(strlists, x))

    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 \*)

(\* a \*)

fun card\_color(suit, rank) =

    case suit of

        Diamonds => Red

        |Hearts => Red

        | \_ => Black

(\* b \*)

fun card\_value(suit, rank) =

    case rank of

        Num n => n

        |Ace => 11

        | \_ => 10

(\* с \*)

fun remove\_card(cs, c, e) =

    let fun help\_fun(cs, acc, isFound) =

        case cs of

            [] => (acc, isFound)

            |(x::xs) => if x = c then (acc @ xs, true) else help\_fun(xs, x::acc, isFound)

    in

     case help\_fun(cs, [], false) of

            (\_, false) => raise e

            | ([], true) => []

            | (x::xs, true) => x::xs

    end

(\* d \*)

fun all\_same\_color [] = true

    |all\_same\_color (x::[]) = true

    |all\_same\_color (x::y::[]) = (card\_color(x) = card\_color(y))

    |all\_same\_color (x::y::xs) = ((card\_color(x) = card\_color(y)) andalso all\_same\_color(xs))

(\* e \*)

fun sum\_cards(cs) =

    let fun help\_fun(cs, sum) =

    case cs of

        [] => sum

        |(x::xs) => help\_fun(xs, sum + card\_value(x))

    in

        help\_fun(cs, 0)

    end

(\* f \*)

fun score(cs, goal) =

    let

        val sum = sum\_cards(cs)

        val previous\_sum = if sum > goal then 3\*(sum - goal) else goal - sum

    in

        if (all\_same\_color(cs)) then

            previous\_sum div 2

        else

            previous\_sum

    end

(\* g \*)

fun officiate(cs, ms, goal) =

    let

    fun current\_state(cs, ms, players\_cards) =

        case ms of

        [] => score(players\_cards, goal)

        |move::moves =>

            case move of

            Discard c => current\_state(cs, moves, remove\_card(players\_cards, c, IllegalMove))

            |Draw =>

                case cs of

                [] => score(players\_cards, goal)

                |x::xs =>

                    if sum\_cards(x::players\_cards) < goal then

                        current\_state(xs, moves, x::players\_cards)

                    else

                        score(x::players\_cards, goal)

    in

    current\_state(cs, ms, [])

    end

**Тести:**

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","Jeffrey"]], "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","Jeffrey"]], "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));

(\* с \*)

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));

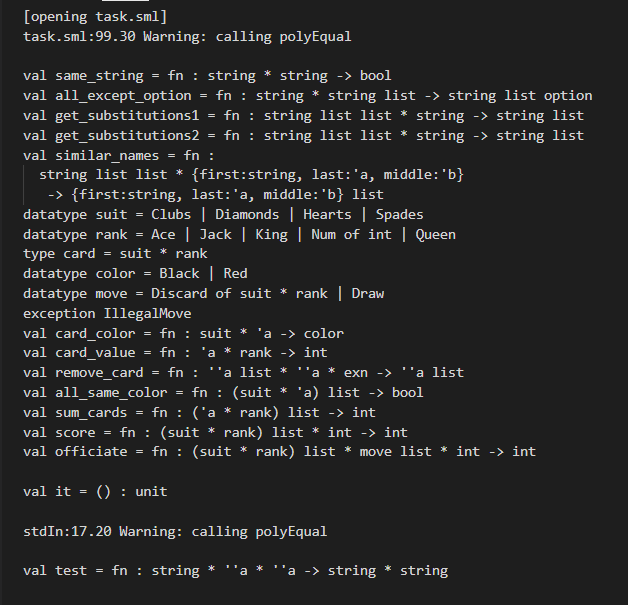
test("score", 2, score(test\_card\_list1, 31));

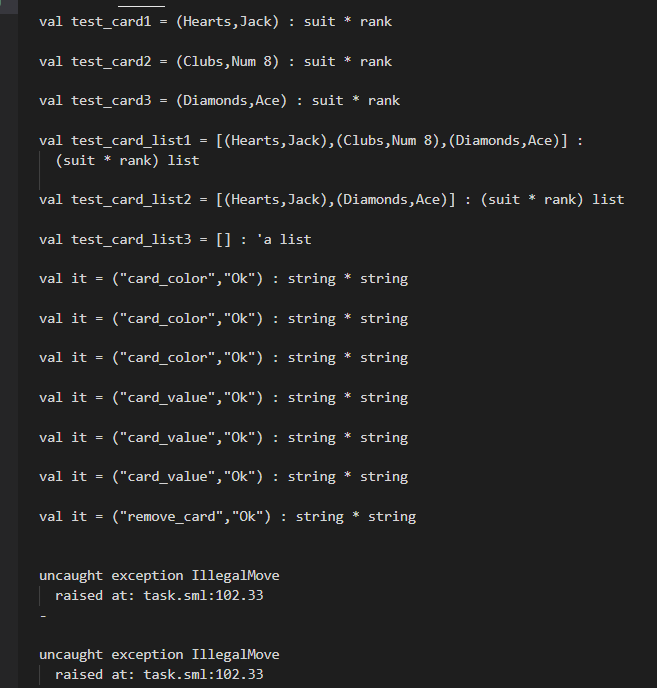
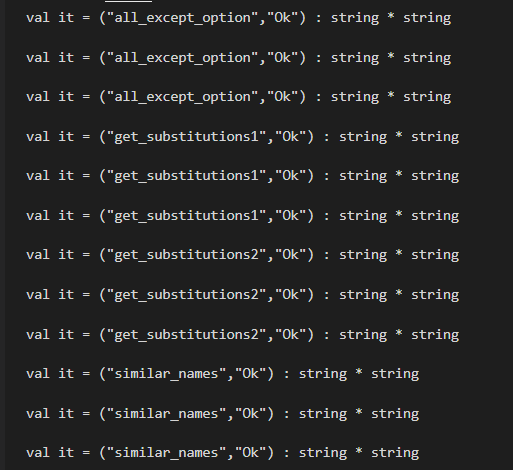
(\* g \*)

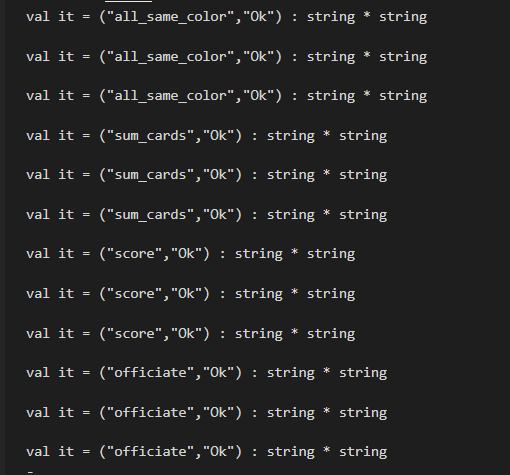
test("officiate", 3, officiate(test\_card\_list1, [Draw, Draw, Draw, Draw], 28));

test("officiate", 5, officiate(test\_card\_list1, [Draw, Draw, Discard(Clubs, Num 8)], 20));

test("officiate", 9, officiate(test\_card\_list1, [Draw, Draw], 15));

****

****

****

Всі функції пройшли всі підготовлені тести