Міністерство освіти і науки України
Національний технічний університет України «Київський політехнічний інститут імені Ігоря Сікорського"
Факультет інформатики та обчислювальної техніки

Кафедра інформатики та програмної інженерії

Звіт

з лабораторної роботи № 3 з дисципліни
«Мультипарадигменне програмування»

Виконав *ІП-01 Заранік Богдан* _____
(шифр, прізвище, ім'я, по батькові)

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

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.

1.1 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. ОПИС ВИКОРИСТАНИХ ТЕХНОЛОГІЙ

Для виконання даної лабораторної роботи мною було використано мову програмування SML, оскільки такою була вимога завдання. У якості середовіща розробки - VSCode та SMLCompiler.

3. ОПИС ПРОГРАМНОГО КОДУ

Task 1A

```
(* 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 *)
    (* PROBLEM 1A *)
fun all_except_option (str, sl) =
  case sl of [] => NONE | x::xs => case same_string(str, x) of
        true => SOME(xs) | false => case all_except_option(str, xs) of
            NONE => NONE | SOME y => SOME(x::y);

fun provided_test1 () =
    let val elem = "zaranik"
        val array = ["zaranik"]
    in
        all_except_option (elem, array) = SOME []
    end;
fun provided_test2 () =
    let val elem = "zaranik"
        val array = ["bogdan"]
    in
        all_except_option (elem, array) = NONE
    end;
fun provided_test3 () =
    let val elem = "zaranik"
        val array = ["zaranik", "barinov", "bazukin"]
    in
        all_except_option (elem, array) = SOME ["barinov", "bazukin"]
    end;
fun provided_test4 () =
    let val elem = "zaranik"
        val array = ["barinov", "zaranik", "bazukin"]
    in
        all_except_option (elem, array) = SOME ["barinov", "bazukin"]
    end;
fun provided_test5 () =
    let val elem = "zaranik"
        val array = ["bazukin", "barinov", "zaranik"]
    in
        all_except_option (elem, array) = SOME ["bazukin", "barinov"]
    end;
val ALL_EXCEPT_OPTION_1 = provided_test1 ()
val ALL_EXCEPT_OPTION_2 = provided_test2 ()
val ALL_EXCEPT_OPTION_3 = provided_test3 ()
val ALL_EXCEPT_OPTION_4 = provided_test4 ()
val ALL_EXCEPT_OPTION_5 = provided_test5 ()
```

Task 1B

```
    (* PROBLEM 1B *)

fun get_substitutions1 (xss, y) =
    case xss of
        [] => [] | xs :: xss' => case all_except_option(y, xs) of
        NONE => get_substitutions1(xss', y) | SOME z => z @ get_substitutions1(xss', y);

fun provided_test1 () =
    let val array = [["foo"],["there"]]
        val elem = "foo"
    in
        get_substitutions1(array, elem) = []
    end;
fun provided_test2 () =
    let val array = [["zaranik","nick"],["don","zaranik","kik"]]
        val elem = "zaranik"
    in
        get_substitutions1(array, elem) = ["nick", "don", "kik"]
    end;
fun provided_test3 () =
    let val array = [["zaranik","nick"],["liza","tania"],["don","zaranik","kik"]]
        val elem = "zaranik"
    in
        get_substitutions1(array, elem) = ["nick","don","kik"]
    end;
fun provided_test4 () =
    let val array =
[["fred","fredrick"],["elizabeth","betty","fred"],["freddie","fred","kik"]]
        val elem = "fred"
    in
        get_substitutions1(array, elem) =
["fredrick","elizabeth","betty","freddie","kik"]
    end;
val GET_SUBSTITUTIONS1_1 = provided_test1 ()
val GET_SUBSTITUTIONS1_2 = provided_test2 ()
val GET_SUBSTITUTIONS1_3 = provided_test3 ()
val GET_SUBSTITUTIONS1_4 = provided_test4 ()
```

Task 1C

```
    (*PROBLEM 1C*)
fun get_substitutions2 (variable1, y) =
        let fun helper (helper_ss, acc) =
            case helper_ss of [] => acc
                | hs :: helper_ss' => case all_except_option(y, hs) of
                    NONE => helper(helper_ss', acc) | SOME z => helper(helper_ss', acc
@ z)
        in helper(variable1, [])
    end;



fun provided_test1 () =
    let val array = [["foo"],["there"]]
        val elem = "foo"
    in
        get_substitutions2(array, elem) = []
    end;
fun provided_test2 () =
    let val array = [["zaranik","nick"],["don","zaranik","kik"]]
        val elem = "zaranik"
    in
        get_substitutions2(array, elem) = ["nick", "don", "kik"]
    end;
fun provided_test3 () =
    let val array = [["zaranik","nick"],["liza","tania"],["don","zaranik","kik"]]
        val elem = "zaranik"
    in
        get_substitutions2(array, elem) = ["nick","don","kik"]
    end;
fun provided_test4 () =
    let val array =
[["fred","fredrick"],["elizabeth","betty","fred"],["freddie","fred","kik"]]
        val elem = "fred"
    in
        get_substitutions2(array, elem) =
["fredrick","elizabeth","betty","freddie","kik"]
    end;
val GET_SUBSTITUTIONS2_1 = provided_test1 ()
val GET_SUBSTITUTIONS2_2 = provided_test2 ()
val GET_SUBSTITUTIONS2_3 = provided_test3 ()
val GET_SUBSTITUTIONS2_4 = provided_test4 ()
```

Task 1D

```
    (*PROBLEM 1D*)
fun similar_names (arrr, {first=f, middle=m, last=l}) =
    let fun function_for_substitution (arr) =
            case arr of [] => [] | x::arr' => {
                first=x,
                middle=m,
                last=l
            } :: function_for_substitution(arr')
    in
        {
            first=f,
            middle=m,
            last=l
        } :: function_for_substitution( get_substitutions2(arrr, f) )
    end;


fun provided_test1 () =
    let val array1 = [
            ["Fred","Fredrick"],
            ["Elizabeth","Betty"],
            ["Freddie","Fred","F"]
        ]
        val full_name = {first="Fred", middle="W", last="Smith"}
        val result_array = [
            {first="Fred", last="Smith", middle="W"},
            {first="Fredrick", last="Smith", middle="W"},
            {first="Freddie", last="Smith", middle="W"},
            {first="F", last="Smith", middle="W"}
        ]
    in
        similar_names(array1, full_name) = result_array
    end;

val SIMILAR_NAMES_1 = provided_test1 ();
```

Task 2A

```
(* 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 *)
    (*PROBLEM 2A*)

fun card_color (suit, rank) =
  case suit of
    Spades => Black
    | Clubs => Black
    | Diamonds => Red
    | Hearts => Red
        fun provided_test1 () =
            let val var1 = Clubs
                val var2 = Num 2
            in
                card_color(var1, var2) = Black
            end;

        fun provided_test2 () =
            let val var1 = Spades
                val var2 = Num 2
            in
                card_color(var1, var2) = Black
            end;

        fun provided_test3 () =
            let val var1 = Diamonds
                val var2 = Num 2
            in
                card_color(var1, var2) = Red
            end;

        fun provided_test4 () =
            let val var1 = Hearts
                val var2 = Num 2
            in
                card_color(var1, var2) = Red
            end;

    val CARD_COLOUR_TEST_1 = provided_test1()
    val CARD_COLOUR_TEST_2 = provided_test2()
    val CARD_COLOUR_TEST_3 = provided_test3()
    val CARD_COLOUR_TEST_4 = provided_test4()
```

Task 2B

```
    (*PROBLEM 2B*)
fun card_value (suit, rank) =
  case rank of
    Jack => 10
    | Queen => 10
    | King => 10
    | Ace => 11
    | Num i => i


        fun provided_test1 () =
            let val var1 = Clubs
                val var2 = Num 2
            in
                card_value(var1, var2) = 2
            end;

        fun provided_test2 () =
            let val var1 = Clubs
                val var2 = Ace
            in
                card_value(var1, var2) = 11
            end;

        fun provided_test3 () =
            let val var1 = Clubs
                val var2 = King
            in
                card_value(var1, var2) = 10
            end;

    val CARD_VALUE_TEST_1 = provided_test1()
    val CARD_VALUE_TEST_2 = provided_test2()
    val CARD_VALUE_TEST_3 = provided_test3()
```

Task 2C

```
    (*PROBLEM 2C*)
fun remove_card (cs : card list, c : card, e : exn) =
    case cs of [] => raise e
        | x::xs => if x=c then xs else x::remove_card(xs, c, e)


        fun provided_test1 () =
            let val var1 = [(Hearts, Ace), (Diamonds, Num 3)]
                val var2 = (Hearts, Ace)
            in
                remove_card(var1, var2, IllegalMove) = [(Diamonds, Num 3)]
            end;

        fun provided_test2 () =
            let val var1 = [(Hearts, Num 2)]
                val var2 = (Hearts, Ace)
            in
                ((remove_card (var1, var2, IllegalMove); false) handle IllegalMove =>
true)
            end;

    val REMOVE_CARD_TEST_1 = provided_test1 ()
    val REMOVE_CARD_TEST_2 = provided_test2 ()
```

Task 2D

```
fun sum_cards (cs : card list) =
    let fun function_helper (xs, accumulate) =
        case xs of
            [] => accumulate | x::xs => function_helper(xs, accumulate + card_value(x))
        in function_helper(cs, 0)
    end

        fun provided_test1 () =
            let val variable = [(Clubs, Num 2),(Clubs, Num 2)]
            in
                sum_cards variable = 4
            end;
        fun provided_test2 () =
            let val variable = [(Clubs, Ace),(Clubs, Num 2)]
            in
                sum_cards variable = 13
            end;
        fun provided_test3 () =
            let val variable = []
            in
                sum_cards variable = 0
            end;
    val SUM_CARDS_TEST_1 = provided_test1()
    val SUM_CARDS_TEST_2 = provided_test2()
    val SUM_CARDS_TEST_3 = provided_test3()
```

Task 2E

```
fun all_same_color (cs : card list) =
    case cs of [] => true
        | _::[] => true
        | head::(neck::rest) => (
            card_color(head) = card_color(neck)
            andalso
            all_same_color(neck::rest)
        );


fun score (cs, goal) =
  let fun pre_score (cs) =
    case (sum_cards(cs), goal) of
        (sum, goal) => case sum > goal of
            true => (sum - goal) * 3
            | false => goal - sum
  in case all_same_color(cs) of
     true => pre_score(cs) div 2
     | false => pre_score(cs)
  End

fun officiate (cards, moves, goal) =
    let
    fun game (cards, helds, []) = score (helds, goal) (* no more moves, game end *)
        | game (cards, helds, x::xs) =
        let
        fun discard_card (cards, helds, card, moves) =
            game (cards, remove_card (helds, card, IllegalMove), moves)
        fun draw_card ([], helds, moves) = score (helds, goal)  (* no more cards, game
end *)
            | draw_card (x::xs, helds, moves) =
            let
            val drawed_helds = x :: helds
            val is_greater = sum_cards (drawed_helds) > goal
            in
            if is_greater then
                (* sum of the helds card exceed the goal, game end *)
                score (drawed_helds, goal)
            else
                game (xs, drawed_helds, moves)
            end
        in
        case x of
            Discard card => discard_card (cards, helds, card, xs)
            | Draw => draw_card (cards, helds, xs)
        end
    in
        game (cards, [], moves)
    end;
```

```
        fun provided_test1 () =
            let val arr1 = [(Hearts, Num 2),(Clubs, Num 4)]
                val arr2 = [Draw]
                val goal = 15
            in
                officiate (arr1,arr2, goal) = 6
            end;

        fun provided_test2 () =
            let val arr1 = [(Clubs,Ace),(Spades,Ace),(Clubs,Ace),(Spades,Ace)]
                val arr2 = [Draw, Draw, Draw, Draw, Draw]
                val goal = 42
            in
                officiate (arr1,arr2, goal) = 3
            end;

        fun provided_test3 () =
            let val arr1 = [(Clubs,Jack),(Spades,Num(8))]
                val arr2 = [Draw,Discard(Hearts,Jack)]
                val goal = 42
            in
                ((officiate(arr1, arr2, goal); false) handle IllegalMove => true)
            end;

    (*TESTS*)
val OFFICIATE_TEST_1 = provided_test1()
val OFFICIATE_TEST_2 = provided_test2()
val OFFICIATE_TEST_3 = provided_test3()
```

4. КРІНШОТИ РОБОТИ ПРОГРАМИ

```
val same_string = fn : string * string -> bool
val all_except_option = fn : string * string list -> string list option
val provided_test1 = fn : unit -> bool
val provided_test2 = fn : unit -> bool
val provided_test3 = fn : unit -> bool
val provided_test4 = fn : unit -> bool
val provided_test5 = fn : unit -> bool
val ALL_EXCEPT_OPTION_1 = true : bool
val ALL_EXCEPT_OPTION_2 = true : bool
val ALL_EXCEPT_OPTION_3 = true : bool
val ALL_EXCEPT_OPTION_4 = true : bool
val ALL_EXCEPT_OPTION_5 = true : bool
val get_substitutions1 = fn : string list list * string -> string list
val provided_test1 = fn : unit -> bool
val provided_test2 = fn : unit -> bool
val provided_test3 = fn : unit -> bool
val provided_test4 = fn : unit -> bool
val GET_SUBSTITUTIONS1_1 = true : bool
val GET_SUBSTITUTIONS1_2 = true : bool
val GET_SUBSTITUTIONS1_3 = true : bool
val GET_SUBSTITUTIONS1_4 = true : bool
val get_substitutions2 = fn : string list list * string -> string list
val provided_test1 = fn : unit -> bool
val provided_test2 = fn : unit -> bool
val provided_test3 = fn : unit -> bool
val provided_test4 = fn : unit -> bool
val GET_SUBSTITUTIONS2_1 = true : bool
val GET_SUBSTITUTIONS2_2 = true : bool
val GET_SUBSTITUTIONS2_3 = true : bool
val GET_SUBSTITUTIONS2_4 = true : bool
```

```
val similar_names = fn
  : string list list * {first:string, last:'a, middle:'b}
    -> {first:string, last:'a, middle:'b} list
val provided_test1 = fn : unit -> bool
val SIMILAR_NAMES_1 = true : bool
datatype suit = Clubs | Diamonds | Hearts | Spades
datatype rank = Ace | Jack | King | Num of int | Queen
type card = suit * rank
datatype color = Black | Red
datatype move = Discard of suit * rank | Draw
exception IllegalMove
val card_color = fn : suit * 'a -> color
val provided_test1 = fn : unit -> bool
val provided_test2 = fn : unit -> bool
val provided_test3 = fn : unit -> bool
val provided_test4 = fn : unit -> bool
val CARD_COLOUR_TEST_1 = true : bool
val CARD_COLOUR_TEST_2 = true : bool
val CARD_COLOUR_TEST_3 = true : bool
val CARD_COLOUR_TEST_4 = true : bool
val card_value = fn : 'a * rank -> int
val provided_test1 = fn : unit -> bool
val provided_test2 = fn : unit -> bool
val provided_test3 = fn : unit -> bool
val CARD_VALUE_TEST_1 = true : bool
val CARD_VALUE_TEST_2 = true : bool
val CARD_VALUE_TEST_3 = true : bool
val remove_card = fn : card list * card * exn -> card list
val provided_test1 = fn : unit -> bool
val provided_test2 = fn : unit -> bool
val REMOVE_CARD_TEST_1 = true : bool
val REMOVE_CARD_TEST_2 = true : bool
```

Програма виконана таким чином, що виконуються усі тести разом, тому скріншот по суті один (лише розбитий на 3 частини) задля покращення читабельності.