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 студент: ІП-01 Танасієнко Олександр

Перевірив: ас. Очеретяний О. К.

Київ 2022

1. **Завдання лабораторної роботи**

Завдання

Практична робота складається із двох завдань. Ваші рішення повинні використовувати відповідність шаблону (pattern-matching). Ви не можете використовувати функції null, hd, tl, isSome або valOf, ви також не можете використовувати будь-що, що містить символ # або функції, які не зазначені в описі лабораторних (наприклад, мутації). Примітка порядок в списках не має значення, якщо конкретно не зазначено в задачі. Завантажте hw02.sml. Наданий код визначає кілька типів для вас. Вам не потрібно буде додавати будь-які додаткові типи даних або синоніми типів. Правильне рішення, без урахування проблемних завдань, становить приблизно 130 рядків, включаючи наданий код.

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. Програмний код

task1.sml

```
fun all_except_option(myStr, stringList) =
    let
        fun traverse(startList, strToFind, leftList) =
            case leftList of
            head::tail => if (head = myStr)
                          then SOME (startList @ tail)
                          else traverse(startList @ (head::[]), myStr, tail)
            |[] => NONE
    in
        traverse([], myStr, stringList)
    end;

fun get_substitutions1(strings, s) =
    case strings of
    head::tail => (case all_except_option(s, head) of
                  NONE => get_substitutions1(tail, s)
                  |SOME lst => lst @ get_substitutions1(tail, s))
    |[] => [];

fun get_substitutions2(strings, s) =
    let
        fun recursive(str, lst, accumulator) =
            case lst of
            head::tail => (case all_except_option(s, head) of
                           NONE => recursive(s, tail, accumulator)
                           |SOME lst => recursive(s, tail, accumulator @ lst))
            |[] => accumulator
    in
        recursive(s, strings, [])
    end;

fun similar_names(strings, {first=f, middle=m, last=l}) =
    let
        fun temp(strs) =
            case strs of
            [] => []
            | head::tail => {first=head, middle=m, last=l}::temp(tail)
    in
        {first=f, middle=m, last=l}::temp(get_substitutions2(strings, f))
    end;
```

test1.sml

```
use "task1.sml";

(*all_except_option tests*)
fun provided_test_all_except_option() =
    let
        val str1="1" val strList1=["1", "4", "5", "6"]
        val str2="5" val strList2=["1", "4", "5", "6"]
        val str3="8" val strList3=["1", "4", "5", "6"]
    in
        [
            all_except_option(str1, strList1),
            all_except_option(str2, strList2),
            all_except_option(str3, strList3)
        ]
    end;

provided_test_all_except_option();

(*get_substitutions1 tests*)
fun provided_test_get_substitutions1() =
    let
        val
strList1=[["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]] val
name1="Fred"
        val
strList2=[["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]] val
name2="Elizabeth"
        val
strList3=[["Fred","Fredrick"],["Elizabeth","Betty"],["John","James","Johny"]] val
name3="John"
    in
        [
            get_substitutions1(strList1, name1),
            get_substitutions1(strList2, name2),
            get_substitutions1(strList3, name3)
        ]
    end;

provided_test_get_substitutions1();

(*get_substitutions2 tests*)
fun provided_test_get_substitutions2() =
    let
        val
strList1=[["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]] val
name1="Fred"
        val
strList2=[["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]] val
name2="Elizabeth"
```

```
        val
strList3=[["Fred","Fredrick"],["Elizabeth","Betty"],["John","James","Johny"]] val
name3="John"
    in
        [
            get_substitutions2(strList1, name1),
            get_substitutions2(strList2, name2),
            get_substitutions2(strList3, name3)
        ]
    end;

provided_test_get_substitutions2();

(*similar_names tests*)
fun provided_test_similar_names() =
    let
        val
strList1=[["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]] val
fullname1={first="Fred", middle="W", last="Smith"}
        val
strList2=[["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]] val
fullname2={first="Betty", middle="F", last="Smith"}
        val
strList3=[["Fred","Fredrick"],["Elizabeth","Betty"],["John","James","Johny"]] val
fullname3={first="John", middle="B", last="Smith"}
    in
        [
            similar_names(strList1, fullname1),
            similar_names(strList2, fullname2),
            similar_names(strList3, fullname3)
        ]
    end;

provided_test_similar_names();
```

task2.sml

```
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

fun card_color(cardSuit, cardRank) =
    case cardSuit of
    Hearts => Red
    | Diamonds => Red
    | _ => Black;
```

```
fun card_value(cardSuit, cardRank) =
    case cardRank of
    Num num => num
    |Ace => 11
    |_ => 10;

fun remove_card(card, cardsList, e) =
    let
        fun remove(checkedCards, cardToFind, leftCards) =
            case leftCards of
            head::tail => (if (head = cardToFind)
                        then checkedCards@tail
                        else remove(checkedCards @ (head::[]), cardToFind, tail))
            |[] => raise e
    in
        remove([], card, cardsList)
    end;

fun all_same_color(cardsList) =
    let
        fun check(lst, checkColor) =
            case lst of
            [] => true
            | head::tail => if card_color(head) <> checkColor
                         then false
                         else check(tail, checkColor)
    in
        case cardsList of
        head::tail => check(tail, card_color(head))
    end;

fun sum_cards(cardsList) =
    let
        fun sum(lst, accum) =
            case lst of
            head::tail => sum(tail, accum+card_value(head))
            |[] => accum;
    in
        sum(cardsList, 0)
    end;

fun score(cardsList, goal) =
    let
        val sum = sum_cards(cardsList)
        fun calcPartialScore() =
            if sum > goal
            then 3*(sum - goal)
            else goal - sum
        val partialScore = calcPartialScore()
    in
        if all_same_color(cardsList) = false
```

```
        then partialScore
        else partialScore div 2
    end;

fun officiate(cardsList, moveList, goal) =
    let
        fun doStep(cardslst, movelst, playerCards) =
            if sum_cards(playerCards) > goal
            then score(playerCards, goal)
            else (
                case movelst of
                step::tailMove => (
                    case step of
                    Draw => (
                        case cardslst of
                        [] => score(playerCards, goal)
                        |cardHead::tailCards => doStep(tailCards, tailMove,
cardHead::playerCards)
                    )
                    |Discard card => doStep(cardslst, tailMove, remove_card(card,
playerCards, IllegalMove))
                )
                |[] => score(playerCards, goal)
            )
    in
        doStep(cardsList, moveList, [])
    end;
```

tests2.sml

```
use "task2.sml";

(*card_color tests*)
fun provided_test_card_color() =
    let
        val card1=(Clubs, Jack)
        val card2=(Diamonds, Queen)
        val card3=(Spades, Num 9)
    in
        [
            card_color(card1),
            card_color(card2),
            card_color(card3)
        ]
    end;

provided_test_card_color();

(*card_value tests*)
fun provided_test_card_value() =
    let
        val card1=(Clubs, Jack)
```

```
        val card2=(Diamonds, Queen)
        val card3=(Spades, Num 9)
    in
        [
            card_value(card1),
            card_value(card2),
            card_value(card3)
        ]
    end;

provided_test_card_value();

(*remove_card tests*)
fun provided_test_remove_card() =
    let
        val card1=(Clubs, Num 10) val cardsList1=[(Clubs, Num 10), (Spades, Num
10), (Diamonds, Num 10)]
        val card2=(Spades, Num 10) val cardsList2=[(Spades, Num 10)]
        val card3=(Diamonds, Num 10) val cardsList3=[(Clubs, Num 10), (Spades,
Num 10), (Diamonds, Num 10)]
    in
        [
            remove_card(card1, cardsList1, IllegalMove),
            remove_card(card2, cardsList2, IllegalMove),
            remove_card(card3, cardsList3, IllegalMove)
        ]
    end;

provided_test_remove_card();

(*all_same_color tests*)
fun provided_test_all_same_color() =
    let
        val cardsList1=[(Clubs, Num 10), (Spades, Num 10), (Diamonds, Num 10)]
        val cardsList2=[(Spades, Num 10)]
        val cardsList3=[(Clubs, Num 10), (Spades, Num 10), (Hearts, Num 10)]
    in
        [
            all_same_color(cardsList1),
            all_same_color(cardsList2),
            all_same_color(cardsList3)
        ]
    end;

provided_test_all_same_color();

(*sum_cards tests*)
fun provided_test_sum_cards() =
    let
        val cardsList1=[(Clubs, Num 10), (Spades, Num 10), (Diamonds, Num 10)]
        val cardsList2=[(Spades, Queen)]
```

```
        val cardsList3=[(Clubs, King), (Spades, Ace), (Hearts, Num 10)]
    in
        [
            sum_cards(cardsList1),
            sum_cards(cardsList2),
            sum_cards(cardsList3)
        ]
    end;

provided_test_sum_cards();

(*score tests*)
fun provided_test_score() =
    let
        val cardsList1=[(Clubs, Num 10), (Spades, Num 10), (Diamonds, Num 10)]
val goal1=50
        val cardsList2=[(Spades, Queen)] val goal2=30
        val cardsList3=[(Clubs, King), (Spades, Ace), (Hearts, Num 10)] val
goal3=20
    in
        [
            score(cardsList1, goal1),
            score(cardsList2, goal2),
            score(cardsList3, goal3)
        ]
    end;

provided_test_score();

(*officiate tests*)
fun provided_test_officiate() =
    let
        val cardsList1=[(Clubs, Num 10), (Spades, Num 10), (Diamonds, Num 10)]
val moves1=[Draw, Draw, Discard (Clubs, Num 10)] val goal1=50
        val cardsList2=[(Spades, Queen)] val moves2=[Draw] val goal2=30
        val cardsList3=[(Clubs, King), (Spades, Ace), (Hearts, Num 10)] val
moves3=[Draw, Discard (Clubs, King), Draw] val goal3=20
    in
        [
            officiate(cardsList1, moves1, goal1),
            officiate(cardsList2, moves2, goal2),
            officiate(cardsList3, moves3, goal3)
        ]
    end;

provided_test_officiate();
```

3. Результати виконання тестів

tests1.sml

```
val provided_test_all_except_option = fn : unit -> string list option list

val it = [SOME ["4","5","6"],SOME ["1","4","6"],NONE] :
  string list option list

val provided_test_get_substitutions1 = fn : unit -> string list list

val it = [["Fredrick","Freddie","F"],["Betty"],["James","Johny"]] :
  string list list

val provided_test_get_substitutions2 = fn : unit -> string list list

val it = [["Fredrick","Freddie","F"],["Betty"],["James","Johny"]] :
  string list list

val provided_test_similar_names = fn :
  unit -> {first:string, last:string, middle:string} list list

val it =
  [[{first="Fred",last="Smith",middle="W"},
    {first="Fredrick",last="Smith",middle="W"},
    {first="Freddie",last="Smith",middle="W"},
    {first="F",last="Smith",middle="W"}],
   [{first="Betty",last="Smith",middle="F"},
    {first="Elizabeth",last="Smith",middle="F"}],
   [{first="John",last="Smith",middle="B"},
    {first="James",last="Smith",middle="B"},
    {first="Johny",last="Smith",middle="B"}]] :
  {first:string, last:string, middle:string} list list
-
```

tests2.sml

```
val provided_test_card_color = fn : unit -> color list

val it = [Black,Red,Black] : color list

val provided_test_card_value = fn : unit -> int list

val it = [10,10,9] : int list

val provided_test_remove_card = fn : unit -> (suit * rank) list list

val it =
  [[(Spades,Num 10),(Diamonds,Num 10)],[],[(Clubs,Num 10),(Spades,Num 10)]] :
  (suit * rank) list list

val provided_test_all_same_color = fn : unit -> bool list

val it = [false,true,false] : bool list

val provided_test_sum_cards = fn : unit -> int list

val it = [30,10,31] : int list

val provided_test_score = fn : unit -> int list

val it = [20,10,33] : int list

val provided_test_officiate = fn : unit -> int list

val it = [20,10,4] : int list
-
```