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(ла) *ІП-01 Пашковський Євгеній Сергійович*
(шифр, прізвище, ім'я, по батькові)

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

Київ 2022

1. Завдання лабораторної роботи

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. Опис використаних технологій

Під час виконання лабораторної роботи було задіяно мову SML (Standard ML) за умовою завдання. Було використано редактор коду Visual Studio Code.

3. 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 *)
(* a *)
fun all_except_option(str : string, str_list : string list) =
   let
     fun hasString(str : string, []) = false |
     hasString(str : string, (x : string)::xs) = same_string(str, x) orelse hasString(str,
xs)
     fun filterByString(str : string, []) = [] |
     filterByString(str : string, (x : string)::xs) = if (same_string(str, x)) then
filterByString(str, xs) else [x] @ filterByString(str, xs)
   in
     if hasString(str, str_list) then SOME(filterByString(str, str_list)) else NONE
   end
(* val a = all_except_option("a", ["b", "a", "c"]) *)
(* b *)
fun get_substitutions1([], s : string) = [] |
get_substitutions1((x : string list)::xs, s : string) =
(case all_except_option(s, x) of
NONE => [] |
SOME (lst : string list) => lst) @ get_substitutions1(xs, s)
(* val b =
get_substitutions1([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]],
"Jeff") *)
(* c *)
fun get_substitutions2(str_list_list : string list list, s : string) =
   let
      fun get_result([], s : string) = [] |
      get_result((x : string list)::xs, s : string) =
      (case all_except_option(s, x) of
      NONE => [] |
      SOME (lst : string list) => lst) @ get_result(xs, s)
   in
     get_result(str_list_list, s)
   end
(* val c =
get_substitutions2([["Fred","Fredrick"],["Jeff","Jeffrey"],["Geoff","Jeff","Jeffrey"]],
"Jeff") *)
(* d *)
fun similar_names(substitutions : string list list, {first:string,middle:string,last:string})
=
   let
     fun get_similar_names([], {first:string,middle:string,last:string}) = [] |
     get_similar_names((x : string)::xs, {first:string,middle:string,last:string}) =
```

```
    [{first=x, middle=middle, last=last}] @ get_similar_names(xs, {first=first,
middle=middle, last=last})
  in
    [{first=first, middle=middle, last=last}] @
get_similar_names(get_substitutions2(substitutions, first), {first=first, middle=middle,
last=last})
  end
(* val d = similar_names([["Fred","Fredrick"],["Elizabeth","Betty"],["Freddie","Fred","F"]],
{first="Fred", middle="W", last="Smith"}) *)
(* 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, _) : card) : color =
  case suit of
  Clubs => Black |
  Spades => Black |
  Diamonds => Red |
  Hearts => Red
(* val a = card_color((Clubs, Jack)) *)
(* b *)
fun card_value((_, rank) : card) : int =
  case rank of
  Ace => 11 |
  King => 10 |
  Queen => 10 |
  Jack => 10 |
  Num(i) => i
(* val b = card_value((Clubs, Num(3))) *)
(* c *)
fun remove_card([], c : card, e : exn) = raise e |
remove_card((x::xs) : card list, c : card, e : exn) : card list =
  if x = c then xs else [x] @ remove_card(xs, c, e)
(* val c = remove_card([(Clubs, Num(3)), (Clubs, Num(3)), (Clubs, Ace)], (Clubs, Num(3)),
IllegalMove) handle IllegalMove => [(Clubs, Num(3)), (Clubs, Num(3)), (Clubs, Ace)] *)
(* d *)
fun all_same_color([]) : bool = true |
all_same_color(x::[]) : bool = true |
all_same_color((x1::x2::xs) : card list) : bool = if card_color(x1) = card_color(x2) then
all_same_color([x2] @ xs) else false
(* val d = all_same_color([(Clubs, Num(3)), (Hearts, Ace), (Spades, Num(2))]) *)
```

```
(* e *)
fun sum_cards([]) : int = 0 |
sum_cards((x::xs) : card list) : int = card_value(x) + sum_cards(xs)
(* val e = sum_cards([(Clubs, Num(3)), (Hearts, Ace), (Spades, Num(2))]); *)
(* f *)
fun score(cl : card list, goal : int) : int =
   let
     val sum = sum_cards(cl)
   in
     (if (sum > goal) then 3 * (sum - goal) else goal - sum) div (if all_same_color(cl) then
2 else 1)
   end
(* val f = score([(Clubs, Num(3)), (Spades, Ace), (Spades, Num(2)), (Spades, Num(6))], 16) *)
(* g *)
fun officiate(cl : card list, ml : move list, goal : int) =
   let
      fun play(deck_list : card list, hand_list : card list, []) = score(hand_list, goal) |
      play(deck_list : card list, hand_list : card list, (move::remaining_moves) : move list)
=
      let
         fun draw([]) = score(hand_list, goal) | draw((x::xs) : card list) =
         if (sum_cards(x::hand_list) > goal) then score(x::hand_list, goal) else play(xs,
x::hand_list, remaining_moves)
         fun discard(c : card) = play(deck_list, remove_card(hand_list, c, IllegalMove),
remaining_moves)
      in
         case move of
         Draw => draw(deck_list) |
         Discard(card) => discard(card)
      end
   in
      play(cl, [], ml)
   end
(* val g = officiate([(Spades, Num(3)), (Clubs, Ace), (Hearts, Num(6))], [Draw,
Discard(Spades, Num(3)), Draw, Draw], 12) *)
```

4. Скріншоти виконання коду

```
- use "lab3.sml";
[opening lab3.sml]
val same_string = fn : string * string -> bool
val all_except_option = fn : string * string list -> string list option
val get_substitutions1 = fn : string list list * string -> string list
val get_substitutions2 = fn : string list list * string -> string list
val similar_names = fn :
  string list list * {first:string, last:string, middle:string}
   -> {first:string, last:string, middle:string} list
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 : card -> color
val card_value = fn : card -> int
val remove_card = fn : card list * card * exn -> card list
val all_same_color = fn : card list -> bool
val sum_cards = fn : card list -> int
val score = fn : card list * int -> int
val officiate = fn : card list * move list * int -> int
val it = () : unit
-
```