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

Кафедра інформатики та програмної інженерії

ЗВІТ

до лабораторних робіт №3

Мультипарадигменне програмування

Виконав студент ІПз01 Максиміхін О.В.
(№ групи, прізвище, ім’я, по батькові)

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

Київ 2022

Завдання.

Завдання 1:

(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:

(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.

Програмний код

lab3func.sml

```
(* Task_1*)
(* 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

(*Task_1a*)
fun all_except_option(str, strList) =
let
    fun find_remove(blankList, str, strList) =
        case strList of
            [] => NONE
            |x::xs' => if same_string(x, str)
                       then SOME (blankList @ xs')
                       else find_remove(blankList @ (x::[]), str, xs')
in
    find_remove([], str, strList)
end;

(*Task_1b*)
fun get_substitutions1(strListList, s) =
    case strListList of
        [] => []
        |x::xs' => (case all_except_option(s, x) of
                        NONE => get_substitutions1(xs', s)
                        |SOME listResult => listResult @ get_substitutions1(xs',
s));

(*Task_1c*)
fun get_substitutions2(strListList, s) =
    let
        fun tailRecursiveCall(lst, str, acc) =
            case lst of
                [] => acc
                |x::xs => (case all_except_option(s, x) of
                              NONE => tailRecursiveCall(xs, s, acc)
                              |SOME listResult => tailRecursiveCall(xs, s, acc @
listResult))
    in
        tailRecursiveCall(strListList, s, [])
    end;


(*Task_1d*)
```

```
fun similar_names(strListList, {first=firstName, middle=middleName, last=lastName})
=
    let
        fun find_mix_result(lst) =
            case lst of
            [] => []
            | x::xs => {first=x, middle=middleName,
last=lastName}::find_mix_result(xs)
    in
        {first=firstName, middle=middleName,
last=lastName}::find_mix_result(get_substitutions2(strListList, firstName))
    end;

(* Task_2*)
(* 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

(*Task_2a*)
fun card_color(card) =
    case card of
    (Diamonds,_) => Red
    | (Hearts,_) => Red
    | (Clubs,_) => Black
    | (Spades,_) => Black;

(*Task_2b*)
fun card_value(card) =
    case card of
    (_,Num num) => num
    |(_,Ace) => 11
    |(_,_) => 10;

(*Task_2c*)
fun remove_card(cs, c, e) =
    let
        fun remove(cardList, card, listResult) =
            case cardList of
                [] => raise e
                |x::xs => (if (x = card)
```

```
                        then listResult @ xs
                        else remove(xs, card, listResult @ (x::[])))
    in
        remove(cs, c, [])
    end;

(*Task_2d*)
fun all_same_color(cs) =
   case cs of
   [] => true
   |x::[] => true
   |(x::xs::xs') => if card_color(x) = card_color(xs)
                    then all_same_color(xs::xs')
                    else false;

(* Task_2e *)
fun sum_cards(cs) =
   let fun sum(cs,acc)=
      case cs of
         []=> acc
         |(x::xs) => sum(xs, acc + card_value(x))
   in
      sum(cs,0)
   end;

(* Task_2f *)
fun score(cs, goal) =
    let fun calculte_score(cs) =
        case sum_cards(cs) > goal of
            true =>  3*(sum_cards(cs) - goal)
            |false =>  goal - sum_cards(cs)
    in
        case all_same_color(cs) of
            false => calculte_score(cs)
            |true => calculte_score(cs) div 2
    end;

(* Task_2g *)
fun officiate(cs, moveList, goal)=
    let fun move_next(cardDeck, moveList, cardsOnHand) =
        case (sum_cards(cardsOnHand) > goal) of
            true => score(cardsOnHand, goal)
            |false => case moveList of
                    [] => score(cardsOnHand, goal)
                    |(headMoveList::tailMoveList) => case headMoveList of
                                Discard card => move_next(cardDeck, tailMoveList,
remove_card(cardsOnHand, card, IllegalMove))
```

```
                                |Draw => case cardDeck of
                                        [] => score(cardsOnHand, goal)
                                        | (headDeck::tailDeck) =>
move_next(tailDeck, tailMoveList, headDeck::cardsOnHand)
   in
      move_next(cs,moveList, [])
   end;
```

Тести

lab3test.sml

```
use ("lab3func.sml");

fun test(fun_name : string, result_true, result_actual) =
if result_true = result_actual
then (fun_name, "Passed")
else (fun_name, "Failed");

print "\n******************* Task_1 ********************\n";
(*Test_Task_1a*)
print "\nTest_Task_1a:\n";
test("all_except_option", SOME ["a", "b", "d", "e"], all_except_option("c", ["a",
"b", "c", "d", "e"]));
test("all_except_option", SOME [], all_except_option("a", ["a"]));
test("all_except_option", NONE, all_except_option("f", ["a", "b", "c", "d", "e"]));
test("all_except_option", NONE, all_except_option("f", []));

(*Test_Task_1b*)
print "\nTest_Task_1b:\n";
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_Task_1c*)
print "\nTest_Task_1c:\n";
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_Task_1d*)
print "\nTest_Task_1d:\n";
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"}));
```

```
print "\n****************** Task_2 ********************\n";
(* Test card variables *)
print "\nVariables:\n";
val card1 = (Diamonds, Jack);
val card2 = (Hearts, Num 10);
val card3 = (Clubs, Ace);
val card4 = (Spades, Num 6);

val list1_all = [card1, card2, card3, card4];
val list2_red = [card1, card2];
val list3_black = [card3, card4];
val list4_blank = [];

(*Test_Task_2a*)
print "\nTest_Task_2a:\n";
test("card_color", Red, card_color(card1));
test("card_color", Red, card_color(card2));
test("card_color", Black, card_color(card3));
test("card_color", Black, card_color(card4));

(*Test_Task_2b*)
print "\nTest_Task_2b:\n";
test("card_value", 10, card_value(card1));
test("card_value", 10, card_value(card2));
test("card_value", 11, card_value(card3));
test("card_value", 6, card_value(card4));

(*Test_Task_2c*)
print "\nTest_Task_2c:\n";
test("remove_card", [card2, card3, card4], remove_card(list1_all, card1,
IllegalMove));
test("remove_card", [card2], remove_card(list2_red, card1, IllegalMove));

(*Test_Task_2d*)
print "\nTest_Task_2d:\n";
test("all_same_color", false, all_same_color(list1_all));
test("all_same_color", true, all_same_color(list2_red));
test("all_same_color", true, all_same_color(list3_black));

(*Test_Task_2e*)
print "\nTest_Task_2e:\n";
test("sum_cards", 37, sum_cards(list1_all));
test("sum_cards", 20, sum_cards(list2_red));
test("sum_cards", 17, sum_cards(list3_black));
test("sum_cards", 0, sum_cards(list4_blank));

(*Test_Task_2f*)
```

```
print "\nTest_Task_2f:\n";
test("score", 30, score(list1_all, 27));
test("score", 5, score(list2_red, 30));
test("score", 0, score(list3_black, 17));

(*Test_Task_2g*)
print "\nTest_Task_2g:\n";
test("officiate", 0, officiate(list1_all, [Draw, Draw, Draw, Draw], 37));
test("officiate", 3, officiate(list1_all, [Draw, Draw, Discard(Diamonds, Jack),
Draw], 20));
test("officiate", 5, officiate(list2_red, [Draw, Draw], 30));
```

Результати виконання тестів

```
********** Task_1 **********
val it = () : unit

Test_Task_1a:
val it = () : unit
val it = ("all_except_option","Passed") : string * string
val it = ("all_except_option","Passed") : string * string
val it = ("all_except_option","Passed") : string * string
val it = ("all_except_option","Passed") : string * string

Test_Task_1b:
val it = () : unit
val it = ("get_substitutions1","Passed") : string * string
val it = ("get_substitutions1","Passed") : string * string

Test_Task_1c:
val it = () : unit
val it = ("get_substitutions2","Passed") : string * string
val it = ("get_substitutions2","Passed") : string * string

Test_Task_1d:
val it = () : unit
val it = ("similar_names","Passed") : string * string

********** Task_2 **********
val it = () : unit
Variables:
val it = () : unit
val card1 = (Diamonds,Jack) : suit * rank
val card2 = (Hearts,Num 10) : suit * rank
val card3 = (Clubs,Ace) : suit * rank
val card4 = (Spades,Num 6) : suit * rank
val list1_all = [(Diamonds,Jack),(Hearts,Num 10),(Clubs,Ace),(Spades,Num 6)] :
  (suit * rank) list
val list2_red = [(Diamonds,Jack),(Hearts,Num 10)] : (suit * rank) list
val list3_black = [(Clubs,Ace),(Spades,Num 6)] : (suit * rank) list
val list4_blank = [] : 'a list

Test_Task_2a:
val it = () : unit
```

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

Test_Task_2b:

```
val it = () : unit
val it = ("card_value","Passed") : string * string
val it = ("card_value","Passed") : string * string
val it = ("card_value","Passed") : string * string
val it = ("card_value","Passed") : string * string
```

Test_Task_2c:

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

Test_Task_2d:

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

Test_Task_2e:

```
val it = () : unit
val it = ("sum_cards","Passed") : string * string
val it = ("sum_cards","Passed") : string * string
val it = ("sum_cards","Passed") : string * string
val it = ("sum_cards","Passed") : string * string
```

Test_Task_2f:

```
val it = () : unit
val it = ("score","Passed") : string * string
val it = ("score","Passed") : string * string
val it = ("score","Passed") : string * string
```

Test_Task_2g:

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