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 Ніколаєв Іван Романович _____
(№ групи, прізвище, ім'я, по батькові)

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

Київ 2022

Завдання :

1. Напишіть функцію only_capitals яка приймає на вхід string list та повертає string list що має тільки рядки що починаються з Великої літери. Вважайте, що всі рядки мають щонайменше один символ. Використайте List.filter, Char.isUpper, та String.sub щоб створити рішення в 1-2 рядки.

2. Напишіть функцію longest_string1 що приймає string list та повертає найдовший string в списку. Якщо список пустий, поверніть "". У випадку наявності декількох однакових кандидатів, поверніть рядок, що найближче до початку списку. Використайте foldl, String.size, та ніякої рекурсії (окрім як використання foldl що є рекурсивним).

3. Напишіть функцію longest_string2 яка точно така сама як longest_string1 окрім як у випадку однакових кандидатів вона повертає найближчого до кінця кандидата. Ваше рішення має бути майже копією longest_string1. Так само використайте foldl та String.size.

4. Напишіть функції longest_string_helper, longest_string3, та longest_string4 такі що:
• longest_string3 має таку саму поведінку як longest_string1 та longest_string4 має таку саму поведінку як longest_string2.
• longest_string_helper має тип (int * int -> bool) -> string list -> string (зверніть увагу на curry). Ця функція буде схожа на longest_string1 та longest_string2 але вона є більш загальною так як приймає функцію як аргумент.
• Якщо longest_string_helper отримує на вхід функцію яка має поведінку як > (тобто повертає true тоді коли перший аргумент строго більше другого), тоді функція має таку саме поведінку як longest_string1.
• longest_string3 та longest_string4 є визначеними через val-прив’язки і часткове використання longest_string_helper.

5. Напишіть функцію longest_capitalized що приймає на вхід string list та повертає найдовший рядок в списку яка починається з Великої літери , або "" якщо таких рядків немає. Вважайте, що всі рядки мають щонайменше один символ. Використовуйте val-прив’язки та ML бібліотечний o оператор для композиції функцій. Вирішіть проблему з однаковими результатами за прикладом завдання 2.

6. Напишіть функцію rev_string, що приймає на вхід string та повертає string що має ті самі символи в зворотньому порядку. Використайте ML o оператор, бібліотечну функцію rev для перевертання списків, та дві бібліотечні функції з String модулю. (Перегляньте документацію, щоб знайти найкращі підходящі)

Наступні дві проблеми передбачають написання функцій над списками які будуть використані в більш пізніх задачах.

7. Напишіть функцію first_answer типу (’a -> ’b option) -> ’a list -> ’b (зауважте 2 аргументи curry). Перший аргумент має бути застосований до елементів другого аргументу до того моменту, як він поверне SOME v для деякого v і тоді v є результатом виклику first_answer. Якщо перший аргумент повертає NONE для всіх елементів списку, тоді має повернути виключення NoAnswer. Підказка: Приклад розв'язку має 5 рядків і не робить нічого складного.

8. Напишіть функцію all_answers типу (’a -> ’b list option) -> ’a list -> ’b list option (зауважте 2 аргументи curry). Перший аргумент має бути застосований до елементів другого аргументу. Якщо результатом є NONE для будь якого з елементів, то результатом all_answers є NONE. Інакше виклики першого аргументу мають повернути SOME lst1, SOME lst2, ... SOME lstn та результатом all_answers буде SOME lst де lst є lst1, lst2, ..., lstn що складаються разом(порядок не важливий).

Підказки: Приклад розв'язку має 8 рядків. Він використовує допоміжні функції з акумулятором та @. Зауважте all_answers f [] має отримати тип SOME [].
Задачі що залишилися використовують наступні визначення типів, що були створені за образом вбудованої реалізації ML порівняння з шаблоном:
datatype pattern = Wildcard | Variable of string | UnitP | ConstP of int | TupleP of pattern list | ConstructorP of string * pattern
datatype valu = Const of int | Unit | Tuple of valu list | Constructor of string * valu

Дано valu v та pattern p, або p співпадає з v або ні. Якщо так,співпадіння створює список string * valu пар; порядок в списку не має значення. Правила порівняння мають бути наступними:
• Wildcard співпадає з усім і створює пустий список прив’язок.
• Variable s співпадає з будь яким значенням v та створює одно елементний список що містить (s,v).
• UnitP співпадає тільки з Unit та створює пустий список прив’язок.
• ConstP 17 співпадає тільки з Const 17 та створює пустий список прив’язок (так само для інших цілих чисел).
• TupleP ps співпадає з значенням форми Tuple vs якщо ps та vs мають однакову довжину і для всіх i, iий елемент ps співпадає з iим елементом vs. Список прив’язок що створюється в результаті є усіма списками вкладених порівнянь з шаблоном що об’єднані в один список.
• ConstructorP(s1,p) співпадає з Constructor(s2,v) якщо s1 та s2 є однаковою строкою (ви можете порівняти їх з =) та p співпадає з v. Список прив’язок

створюється із вкладених порівнянь із шаблоном. Ми називаємо рядки s1 та s2 іменами конструкторів.
• Все інше не має значення.

9. (Ця задача використовує pattern тип даних але не зовсім про порівняння із шаблоном.) Функція g надана в файлі.
(1) Використайте g для визначення функції count_wildcards, що приймає на вхід pattern та повертає скільки Wildcard pattern-ів він містить.
(2) Використайте g для визначення функції count_wild_and_variable_lengths що приймає на вхід pattern та повертає кількість Wildcard pattern-ів які він містить плюс суму довжин рядків всіх змінних що містяться у змінній patterns. (Використайте String.size. Нам важливі тільки імена змінних; імена конструкторів не важливі.)
(3) Використайте g для визначення функції count_some_var що приймає на вхід строку та pattern (як пару) та повертає кількість входжень строки як змінної в pattern. Нам важливі тільки імена змінних; імена конструкторів не важливі.

10. Напишіть функцію check_pat що приймає на вхід pattern та повертає true тоді і тільки тоді коли всі змінні що з'являються в pattern відрізняються один від одного (наприклад, використовують різні рядки). Імена конструкторів не важливі. Підказки: Приклад розв'язку має 2 допоміжні функції. Перша приймає pattern та повертає список всіх рядків які він використовує для змінних. Використовуючи foldl з функцією яка використовує append може бути корисним. Друга функція приймає на вхід список рядків і вирішує чи він має повтори. List.exists може бути корисним. Приклад розв'язку має 15 рядків. Підказка: foldl та List.exists не обов'язкові, але можуть допомогти.

11. Напишіть функцію first_match що приймає на вхід value та список шаблонів та повертає (string * valu) list option, тобто NONE якщо ніякий паттерн зі списку не підходить або SOME lst де lst це список прив'язок для першого паттерну в списку який підійшов. Використайте first_answer та handle-вираз. Підказка: Приклад розв'язку має 3 рядки.

Програмний код

(task.sml)

```
val only_capitals = List.filter (fn str => (Char.isUpper o String.sub) (str, 0))

val longest_string1 =
    List.foldl (fn (str, acc) =>
           if (String.size str) > (String.size acc)
           then str
           else acc) ""


val longest_string2 =
```

```
    List.foldl (fn (str, acc) =>
           if (String.size str) >= (String.size acc)
           then str
           else acc) ""

fun longest_string_helper f =
    List.foldl (fn (str, acc) =>
           if f (String.size str, String.size acc)
           then str
           else acc) ""

val longest_string3 = longest_string_helper (fn (a, b) => a > b)

val longest_string4 = longest_string_helper (fn (a, b) => a >= b)

val longest_capitalized = longest_string1 o only_capitals

val rev_string = String.implode o rev o String.explode

exception NoAnswer

fun first_answer f lst =
    case lst of
    [] => raise NoAnswer
      | x :: xs => case f x of
                SOME v => v
              | NONE => first_answer f xs

fun all_answers f lst =
    let
    fun all_answers_helper remaining acc =
        case (remaining, acc) of
        ([], _) => acc
          | (x :: xs, SOME v) => (case f x of
                       NONE => NONE
                     | SOME xv => all_answers_helper xs (SOME (xv @ v)))
          | _ => NONE
    in
    all_answers_helper lst (SOME [])
    end

datatype pattern = Wildcard | Variable of string | UnitP | ConstP of int | TupleP
of pattern list | ConstructorP of string * pattern
datatype valu = Const of int | Unit | Tuple of valu list | Constructor of string
* valu


fun g f1 f2 p =
    let
    val r = g f1 f2
    in
```

```
    case p of
        Wildcard          => f1 ()
      | Variable x        => f2 x
      | TupleP ps         => List.foldl (fn (p,i) => (r p) + i) 0 ps
      | ConstructorP(_,p) => r p
      | _                 => 0
    end

val count_wildcards = g (fn _ => 1) (fn _ => 0)

val count_wild_and_variable_lengths = g (fn _ => 1) String.size

fun count_some_var (str, p) = g (fn _ => 0) (fn x =>
                        if String.isSubstring str x
                        then 1
                        else 0) p

fun check_pat p =
    let
    fun filterString pat acc = case pat of
                        Variable x => x :: acc
                      | ConstructorP (_, p) => filterString p acc
                      | TupleP ps =>
                        List.foldl
                        (fn (p, acc) => (filterString p []) @ acc) [] ps
                      | _ => []
    in
    let
        val strList = filterString p []
        fun checkDuplicate remList =
        case remList of
            [] => true
          | x :: xs => if List.exists (fn item => item = x) xs
                    then false
                    else checkDuplicate xs
    in
        checkDuplicate strList
    end
    end

fun match (v, p) =
    case p of
    Wildcard => SOME []
      | UnitP => (case v of Unit => SOME []
                | _ => NONE)
      | Variable str => SOME [(str, v)]
      | ConstP i => (case v of Const j => if i = j then SOME [] else NONE
                  | _ => NONE)
      | TupleP plst => (case v of
                Tuple vlst => if List.length plst = List.length vlst
                      then all_answers match (ListPair.zip (vlst, plst))
```

```
                    else NONE
              | _ => NONE)
      | ConstructorP (str, pt) => (case v of
                    Constructor (vstr, vval) => if str = vstr
                                then match (vval, pt)
                                else NONE
                  | _ => NONE)

fun first_match v plst =
    SOME (first_answer (fn p => match (v, p)) plst)
    handle NoAnswer => NONE
```

Тести

(test.sml)

```
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");
;

test("only_capitals", ["Bx"], only_capitals(["abc", "aXa", "Bx"]));
test("only_capitals", [], only_capitals(["abc", "aXa", "kx"]));

test("longest_string1", "", longest_string1([]));
test("longest_string1", "abcde", longest_string1(["abcde", "1234", "kx"]));
test("longest_string1", "abcde", longest_string1(["abcde", "12345", "kx"]));

test("longest_string2", "", longest_string2([]));
test("longest_string2", "abcde", longest_string2(["abcde", "1234", "kx"]));
test("longest_string2", "12345", longest_string2(["abcde", "12345", "kx"]));

test("longest_string3", "", longest_string3([]));
test("longest_string3", "abcde", longest_string3(["abcde", "1234", "kx"]));
test("longest_string3", "abcde", longest_string3(["abcde", "12345", "kx"]));


test("longest_string4", "", longest_string4([]));
test("longest_string4", "abcde", longest_string4(["abcde", "1234", "kx"]));
test("longest_string4", "12345", longest_string4(["abcde", "12345", "kx"]));

test("longest_capitalized", "", longest_capitalized([]));
test("longest_capitalized", "Abcde", longest_capitalized(["Abcde", "1234",
"kx"]));
test("longest_capitalized", "", longest_capitalized(["abcde", "12345", "kx"]));
```

```
test("rev_string", "", rev_string(""));
test("rev_string", "edcbA", rev_string("Abcde"));

test("first_answer", 5, first_answer (fn(x) => if x = 5 then SOME 5 else
NONE)  [1,2,3,4,5]);
test("first_answer", 0, first_answer (fn(x) => if x = 6 then SOME 6 else
NONE)  [1,2,3,4,5]);

test("all_answers", SOME [], all_answers (fn(x) => if x = 5 then SOME [5] else
NONE)  []);
test("all_answers", NONE, all_answers (fn(x) => if x = 5 then SOME [5] else
NONE)  [1,2,3,4,5]);
test("all_answers", SOME [5], all_answers (fn(x) => if x = 5 then SOME [5] else
NONE)  [5]);
test("all_answers", SOME [5,5], all_answers (fn(x) => if x = 5 then SOME [5] else
NONE)  [5,5]);

test("count_wildcards", 1 ,count_wildcards Wildcard);
test("count_wildcards", 1,count_wildcards (TupleP ([Wildcard])));
test("count_wildcards", 2,count_wildcards (TupleP ([Wildcard,Wildcard])));
test("count_wildcards", 1,count_wildcards (TupleP ([Wildcard,ConstP(1)])));
test("count_wildcards", 3,count_wildcards (TupleP
([Wildcard,Wildcard,Wildcard,ConstP(1)])));
test("count_wildcards", 0,count_wildcards (TupleP ([ConstP(1),ConstP(1)])));

test("count_wild_and_variable_lengths", 3,count_wild_and_variable_lengths (TupleP
([Wildcard,Wildcard,Wildcard])));
test("count_wild_and_variable_lengths", 4,count_wild_and_variable_lengths (TupleP
([Wildcard,Wildcard,Variable("ab")])));
test("count_wild_and_variable_lengths", 2,count_wild_and_variable_lengths (TupleP
([Variable("ab")])));
test("count_wild_and_variable_lengths", 9,count_wild_and_variable_lengths (TupleP
([Variable("ab"),Variable"abcde",Wildcard,Wildcard])));

test("count_some_var", 1,count_some_var("ab", (TupleP ([Variable("ab")]))));
test("count_some_var", 2,count_some_var("ab", (TupleP
([Variable("ab"),Variable("bc"),Variable("ab")]))));
test("count_some_var", 0,count_some_var("ab2", (TupleP
([Variable("ab"),Variable("bc"),Variable("ab")]))));
test("count_some_var", 0,count_some_var("wild",ConstructorP
("wild",(Wildcard))));
test("count_some_var", 2,count_some_var ("x",TupleP[TupleP[TupleP[Variable
"x",ConstructorP("wild",Wildcard)],Wildcard],Variable "x"]));

test("check_pat", true, check_pat((TupleP ([Variable("")]))));
test("check_pat", true, check_pat((TupleP ([Variable("ab")]))));
test("check_pat", false, check_pat((TupleP
([Variable("ab"),Variable("ab"),Variable("bc")]))));
test("check_pat", true, check_pat( (TupleP
([Variable("ab1"),Variable("ab2"),Variable("ab3")]))));
```

```
test("match", NONE, match( Const 1,TupleP ([Variable("")])));
test("match", SOME [], match( Const 1,ConstP 1));
test("match", SOME[], match( Tuple[ (Const 1), Unit,Constructor("asd", Const 1),
Constructor("dsa",Const 2)],
                            TupleP ([ConstP 1,UnitP,ConstructorP ("asd", ConstP
1),ConstructorP("dsa",ConstP 2) ])));
test("match", SOME [("dsa3",Constructor("dsa",Const
2)),("dsa2",Constructor("asd", Const 1)),
                                        ("dsa1",Const 1)], match( Tuple[
(Const 1), Unit,Constructor("asd", Const 1), Constructor("dsa",Const 2)],
                            TupleP ([Variable("dsa1"), UnitP , Variable("dsa2")
,Variable("dsa3")])));

test("first_match", NONE, first_match (Constructor("dsa",Const 1)) ([UnitP,ConstP
1]));
test("first_match", SOME [("abc",Const 1)], first_match (Const 1)
([UnitP,Variable("abc")]));
test("first_match", SOME [("abc",Const 1)], first_match (Const 1)
([Variable("abc"),Variable("abc1")]));
```

```
val it = ("only_capitals","Ok") : string * string

val it = ("only_capitals","Ok") : string * string

val it = ("longest_string1","Ok") : string * string

val it = ("longest_string1","Ok") : string * string

val it = ("longest_string1","Ok") : string * string

val it = ("longest_string2","Ok") : string * string

val it = ("longest_string2","Ok") : string * string

val it = ("longest_string2","Ok") : string * string

val it = ("longest_string3","Ok") : string * string

val it = ("longest_string3","Ok") : string * string

val it = ("longest_string3","Ok") : string * string

val it = ("longest_string4","Ok") : string * string
```

```
val it = ("longest_capitalized","Ok") : string * string

val it = ("longest_capitalized","Ok") : string * string

val it = ("longest_capitalized","Ok") : string * string

val it = ("rev_string","Ok") : string * string

val it = ("rev_string","Ok") : string * string

val it = ("first_answer","Ok") : string * string
-
val it = ("all_answers","Ok") : string * string

val it = ("all_answers","Ok") : string * string

val it = ("all_answers","Ok") : string * string

val it = ("all_answers","Ok") : string * string

val it = ("count_wildcards","Ok") : string * string

val it = ("count_wildcards","Ok") : string * string

val it = ("count_wildcards","Ok") : string * string

val it = ("count_wildcards","Ok") : string * string

val it = ("count_wildcards","Ok") : string * string
```

```
val it = ("count_wild_and_variable_lengths","Ok") : string * string

val it = ("count_wild_and_variable_lengths","Ok") : string * string

val it = ("count_wild_and_variable_lengths","Ok") : string * string

val it = ("count_some_var","Ok") : string * string

val it = ("count_some_var","Ok") : string * string

val it = ("count_some_var","Ok") : string * string

val it = ("count_some_var","Ok") : string * string

val it = ("count_some_var","Ok") : string * string

val it = ("check_pat","Ok") : string * string

val it = ("check_pat","Ok") : string * string

val it = ("check_pat","Ok") : string * string

val it = ("check_pat","Ok") : string * string

val it = ("match","Ok") : string * string

val it = ("match","Ok") : string * string

val it = ("match","Ok") : string * string
```

```
val it = ("first_match","Ok") : string * string

val it = ("first_match","Ok") : string * string

val it = ("first_match","Ok") : string * string
-
```