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  № 4 з дисципліни**

**“Мультипарадигменне програмування”**

Виконав студент: ІП-01 Танасієнко Олександр

Перевірив: ас. Очеретяний О. К.

Київ 2022

1. **Завдання лабораторної роботи**

## 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**](https://drive.google.com/file/d/1PdENR1ivx_ng6dIakZEG9456xWn-lkpQ/view?usp=sharing).

(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.

1. **Програмний код**

**functions.sml**

fun only\_capitals(*stringList*: string list) =

    List.filter (fn *str* =>

                    Char.isUpper(String.sub(str, 0)))

                stringList;

fun longest\_string1(*stringList*: string list) =

    List.foldl (fn (*str1*, *str2*) =>

                    if (String.size str1) > (String.size str2)

                    then str1

                    else str2)

                ""

                stringList;

fun longest\_string2(*stringList*: string list) =

    List.foldl (fn (*str1*, *str2*) =>

                    if (String.size str1) >= (String.size str2)

                    then str1

                    else str2)

                ""

                stringList;

fun longest\_string\_helper *stringComparator* *stringList* =

    List.foldl (fn (*str1*, *str2*) =>

                    if stringComparator(String.size str1, String.size str2)

                    then str1

                    else str2)

                ""

                stringList;

val longest\_string3 = longest\_string\_helper (fn (*str1*, *str2*) => str1 > str2);

val longest\_string4 = longest\_string\_helper (fn (*str1*, *str2*) => str1 >= str2);

val longest\_capitalized = longest\_string1 o only\_capitals;

val rev\_string = String.implode o rev o String.explode;

exception NoAnswer;

fun first\_answer *functionToApply* *list* =

    case list of

*head*::*tail* => (case functionToApply(head) of

                   SOME *v* => v

                   |NONE => (first\_answer functionToApply tail))

    |[] => raise NoAnswer;

fun all\_answers *functionToApply* *list* =

    let

        fun helper(*tl*, *accum*) =

            case tl of

*head*::*tail* => (case functionToApply(head) of

                           SOME *v* => helper(tail, accum @ v)

                           |NONE => NONE)

            |[] => SOME (accum)

    in

        helper(list, [])

    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;

fun count\_wildcards(pat: pattern) =

    g (fn \_ => 1)

      (fn \_ => 0)

      pat;

fun count\_wild\_and\_variable\_lengths(pat: pattern) =

    g (fn \_ => 1)

      (String.size)

      pat;

fun count\_some\_var(str: string, pat: pattern) =

    g (fn \_ => 0)

      (fn *strParam* => if strParam = str

                      then 1

                      else 0)

      pat;

fun check\_pat(pat: pattern) =

    let

        fun getStrVariables(*patt*: pattern) =

            case patt of

            Variable *x* => [x]

            |TupleP *ps* => List.foldl(fn (*p1*, *acc*) =>

                                        getStrVariables(p1) @ acc)

                                     []

                                     ps

            |ConstructorP(\_,*p*) => getStrVariables(p)

            |\_ => []

        fun checkDublicates(*stringList*: string list) =

            case stringList of

            [] => true

            |*head*::*tail* => if (List.exists (fn *s* => s = head) tail)

                           then false

                           else checkDublicates(tail)

    in

        checkDublicates(getStrVariables(pat))

    end;

fun first\_match(someValue: valu, patternList: pattern list) =

    let

        fun matchHelper(*v*: valu, somePattern: pattern) =

            case (v, somePattern) of

            (\_, Wildcard) => SOME []

            |(\_, Variable *s*) => SOME [(s,v)]

            |(Unit, UnitP) => SOME []

            |(Const *vLocal*, ConstP *p*) => (if vLocal = p

                                          then SOME []

                                          else NONE)

            |(Tuple *vs*, TupleP *ps*) => (if (List.length vs) = (List.length ps)

                                       then all\_answers matchHelper (ListPair.zip(vs, ps))

                                       else NONE)

            |(Constructor (*s1*, *vLocal*), ConstructorP (*s2*, *pLocal*)) => (if s1 = s2

                                                                       then matchHelper(vLocal, pLocal)

                                                                       else NONE)

            |(\_, \_) => NONE

    in

        SOME (first\_answer (fn *patParam* => matchHelper(someValue, patParam)) patternList) handle NoAnswer => NONE

    end;

**tests.sml**

use "functions.sml";

(\*only\_capitals tests\*)

fun provided\_test\_only\_capitals() =

    let

        val strList1 = ["Fred", "Fredrick", "bicycle", "Elizabeth", "Betty", "Freddie", "person"]

        val strList2 = ["London","England", "Poland"]

        val strList3 = ["car", "plane", "airport", "ship"]

    in

        [

            only\_capitals(strList1),

            only\_capitals(strList2),

            only\_capitals(strList3)

        ]

    end;

provided\_test\_only\_capitals();

(\*longest\_string1 tests\*)

fun provided\_test\_longest\_string1() =

    let

        val strList1 = []

        val strList2 = ["1111","2222", "3333"]

        val strList3 = ["car", "aircraft", "airport", "ship", "bicylcle"]

    in

        [

            longest\_string1(strList1),

            longest\_string1(strList2),

            longest\_string1(strList3)

        ]

    end;

provided\_test\_longest\_string1();

(\*longest\_string2 tests\*)

fun provided\_test\_longest\_string2() =

    let

        val strList1 = []

        val strList2 = ["1111","2222", "3333"]

        val strList3 = ["car", "aircraft", "airport", "ship", "bicylcle"]

    in

        [

            longest\_string2(strList1),

            longest\_string2(strList2),

            longest\_string2(strList3)

        ]

    end;

provided\_test\_longest\_string2();

(\*longest\_string3 tests\*)

fun provided\_test\_longest\_string3() =

    let

        val strList1 = []

        val strList2 = ["1111","2222", "3333"]

        val strList3 = ["car", "aircraft", "airport", "ship", "bicylcle"]

    in

        [

            longest\_string3(strList1),

            longest\_string3(strList2),

            longest\_string3(strList3)

        ]

    end;

provided\_test\_longest\_string3();

(\*longest\_string4 tests\*)

fun provided\_test\_longest\_string4() =

    let

        val strList1 = []

        val strList2 = ["1111","2222", "3333"]

        val strList3 = ["car", "aircraft", "airport", "ship", "bicylcle"]

    in

        [

            longest\_string4(strList1),

            longest\_string4(strList2),

            longest\_string4(strList3)

        ]

    end;

provided\_test\_longest\_string4();

(\*longest\_capitalized tests\*)

fun provided\_test\_longest\_capitalized() =

    let

        val strList1 = ["veryLongWord", "Fred", "Frederick", "bicycle", "Elizabeth", "Betty", "Freddie", "person"]

        val strList2 = ["London","England", "Poland"]

        val strList3 = ["car", "plane", "airport", "ship"]

    in

        [

            longest\_capitalized(strList1),

            longest\_capitalized(strList2),

            longest\_capitalized(strList3)

        ]

    end;

provided\_test\_longest\_capitalized();

(\*longest\_capitalized tests\*)

fun provided\_test\_rev\_string() =

    let

        val str1 = "Elizabeth"

        val str2 = "London"

        val str3 = "airport"

    in

        [

            rev\_string(str1),

            rev\_string(str2),

            rev\_string(str3)

        ]

    end;

provided\_test\_rev\_string();

(\*first\_answer tests\*)

fun provided\_test\_first\_answer() =

    let

        val list1 = [5, 4, 3, 2, 1]

        val list2 = [4]

        val list3 = [1, 2, 3]

    in

        [

            (if (first\_answer (fn *arg* => if arg = 4 then SOME (arg) else NONE) list1) = 4 then "Right" else "NOT") handle (NoAnswer) => "Wrong",

            (if (first\_answer (fn *arg* => if arg = 4 then SOME (arg) else NONE) list2) = 4 then "Right" else "NOT") handle (NoAnswer) => "Wrong",

            (if (first\_answer (fn *arg* => if arg = 4 then SOME (arg) else NONE) list3) = 4 then "Right" else "NOT") handle (NoAnswer) => "Wrong"

        ]

    end;

provided\_test\_first\_answer();

(\*all\_answers tests\*)

fun provided\_test\_all\_answers() =

    let

        val list1 = [5, 4, 3, 2, 1]

        val list2 = [4]

        val list3 = [1, 2, 3]

    in

        [

            all\_answers (fn *arg* => if arg > 0 then SOME [arg] else NONE) list1,

            all\_answers (fn *arg* => if arg = 4 then SOME [arg] else NONE) list2,

            all\_answers (fn *arg* => if arg < 3 then SOME [arg] else NONE) list3

        ]

    end;

provided\_test\_all\_answers();

(\*count\_wildcards tests\*)

fun provided\_test\_count\_wildcards() =

    let

        val pattern1: pattern = (TupleP ([ConstP(1),ConstP(1), ConstructorP("mystr", Wildcard)]))

        val pattern2: pattern = Wildcard

        val pattern3: pattern = (TupleP ([Wildcard,ConstP(1), Wildcard, Wildcard]))

        val pattern4: pattern = Variable "hello"

    in

        [

            count\_wildcards(pattern1),

            count\_wildcards(pattern2),

            count\_wildcards(pattern3),

            count\_wildcards(pattern4)

        ]

    end;

provided\_test\_count\_wildcards();

(\*count\_wild\_and\_variable\_lengths tests\*)

fun provided\_test\_count\_wild\_and\_variable\_lengths() =

    let

        val pattern1: pattern = (TupleP ([Variable "car", Variable "bicycle", ConstructorP("mystr", Wildcard)]))

        val pattern2: pattern = Wildcard

        val pattern3: pattern = (TupleP ([Wildcard, Variable "sometext", Wildcard, Wildcard]))

        val pattern4: pattern = Variable "hello"

    in

        [

            count\_wild\_and\_variable\_lengths(pattern1),

            count\_wild\_and\_variable\_lengths(pattern2),

            count\_wild\_and\_variable\_lengths(pattern3),

            count\_wild\_and\_variable\_lengths(pattern4)

        ]

    end;

provided\_test\_count\_wild\_and\_variable\_lengths();

(\*count\_some\_var tests\*)

fun provided\_test\_count\_some\_var() =

    let

        val str1 = "car" val pattern1: pattern = (TupleP ([Variable "car", Variable "car", ConstructorP("mystr", Wildcard)]))

        val str2 = "car"  val pattern2: pattern = Wildcard

        val str3 = "sometext"  val pattern3: pattern = (TupleP ([Wildcard, Variable "sometext", Wildcard, Wildcard]))

        val str4 = "hello"  val pattern4: pattern = Variable "hello"

    in

        [

            count\_some\_var(str1, pattern1),

            count\_some\_var(str2, pattern2),

            count\_some\_var(str3, pattern3),

            count\_some\_var(str4, pattern4)

        ]

    end;

provided\_test\_count\_some\_var();

(\*check\_pat tests\*)

fun provided\_test\_check\_pat() =

    let

        val pattern1: pattern = (TupleP ([Variable "car", Variable "car", ConstructorP("mystr", Wildcard)]))

        val pattern2: pattern = Wildcard

        val pattern3: pattern = (TupleP ([Wildcard, Variable "sometext", Wildcard, Wildcard, Variable "another text"]))

        val pattern4: pattern = Variable "hello"

    in

        [

            check\_pat(pattern1),

            check\_pat(pattern2),

            check\_pat(pattern3),

            check\_pat(pattern4)

        ]

    end;

provided\_test\_check\_pat();

(\*first\_match tests\*)

fun provided\_test\_first\_match() =

    let

        val valu1: valu = Const 5

        val pattern1: pattern list = [Variable "car", ConstP 5, ConstructorP("mystr", Wildcard)]

        val valu2: valu = Unit

        val pattern2: pattern list = [UnitP]

        val valu3: valu = Constructor("str1", Const 1)

        val pattern3: pattern list = [Wildcard, Variable "sometext", ConstructorP("str1", ConstP 1)]

        val valu4: valu = Tuple ([Const 5, Const 1])

        val pattern4: pattern list = [TupleP ([ConstP 5, Variable "car"])]

    in

        [

            first\_match(valu1, pattern1),

            first\_match(valu2, pattern2),

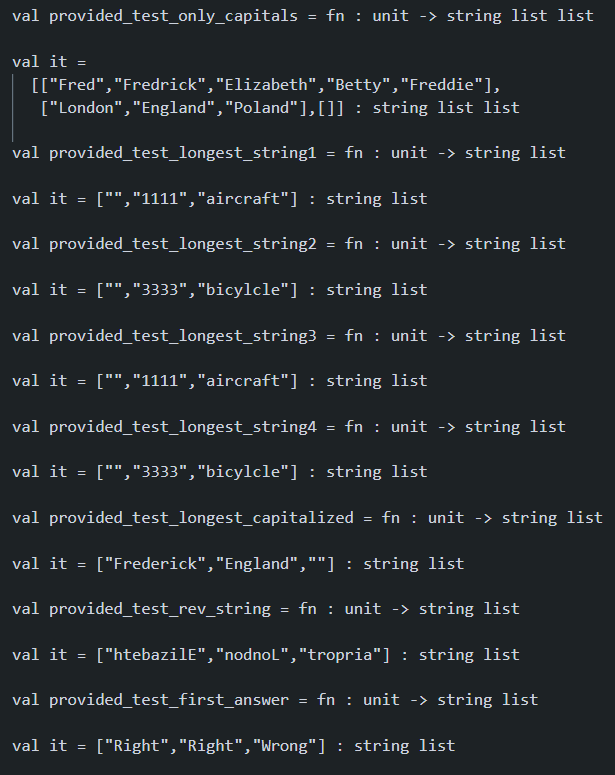
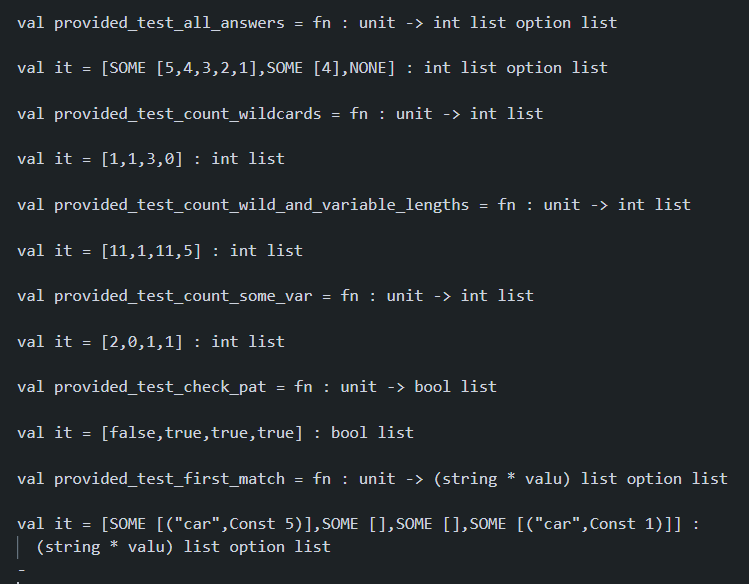
            first\_match(valu3, pattern3),

            first\_match(valu4, pattern4)

        ]

    end;

provided\_test\_first\_match();

1. **Результати виконання тестів** ****