Одного поля ягоды

Язык: F#

Максимальное количество баллов: (не считая бонусной задачи): 14

Большинство функций, которые нужно написать в этой работе, получатся короткими. Используйте функции высшего порядка, композицию, конвейеры, функции стандартной библиотеки. Типы данных и кое-какой код предоставлены в файле hw3_provided.fs. Эталонное решение, не считая бонусной задачи, содержит примерно 110 строк кода.

1. (+1) Напишите функцию **`only_capitals`**, которая принимает **`string list`** и возвращает **`string list`**, который содержит только те строки из исходного списка, которые начинаются с заглавной буквы. Считайте, что все строки содержат не меньше одного символа. Используйте подходящую функцию высшего порядка из стандартной библиотеки, а так же **`Char.IsUpper`**.
2. (+1) Напишите функцию **`longest_string1`**, которая принимает **`string list`** и возвращает наиболее длинную строку из исходного списка. Если исходный список пуст, возвращайте **`""`**. Если таких строк несколько возвращайте ту, которая ближе к началу списка. Воспользуйтесь **`List.fold`** и **`String.length`**.
3. (+1) Напишите функцию **`longest_string2`**, которая ведёт себя аналогично предыдущей, но в случае равных по длине строк возвращает строку, которая ближе к концу списка.
4. (+1) Напишите функции **`longest_string_helper`**, **`longest_string3`** и **`longest_string4`**:
    a. Поведение **`longest_string3`** аналогично **longest_string1**, поведение **`longest_string4`** аналогично **`longest_string2`**
    b. **`longest_sting_helper`** имеет тип (например) **`(int -> int -> bool) -> (System.String list -> System.String)`**
    Она похожа на **`longest_string1`** и **`longest_string2`**, только более универсальна.

    c. Если передать функции **longest_string_helper** функцию, которая ведёт себя как **>**, то результирующая функция будет вести себя аналогично **longest_string1**
    d. **longest_string3** и **longest_string4** определите с помощью частичного применения **longest_string_helper.**
5. (+1) Напишите функцию **longest_capitalized**, которая принимает **string list** и возвращает самую длинную строку из тех, которые начинаются с заглавной буквы, или **""**, если таких строк нет. Воспользуйтесь оператором композиции функций. В случае неоднозначности, поступайте так же, как в задаче 2.
6. (+1) Напишите функцию **rev_string**, которая принимает и переворачивает строку. Воспользуйтесь подходящими функциями из стандартной библиотеки и оператором конвейера.
7. (+1) Напишите функцию **first_answer** типа **('a -> 'b option) -> 'a list -> 'b**. Первый аргумент должен последовательно вызываться с элементами второго в качестве аргумента, до тех пор, пока не будет получен **Some v**, тогда **v** - результат вызова **first_answer**. Если первый аргумент возвращает **None** для всех элементов, выбросьте исключение **NoAnswer**.
8. (+1) Напишите функцию **all_answers** типа **('a -> 'b list option) -> 'a list -> 'b list option**. Первый аргумент должен последовательно вызываться с элементами второго в качестве аргумента. Если он вернёт **None** хотя бы для одного элемента, результат **all_answers** должен быть **None**. В противном случае, если вызовы первого аргумента вернут **Some lst1, Some lst2** … **Some lstn**, то результат **all_answers** должен представлять собой **Some lst**, где **lst** - это **lst1, lst2** … **lstn**, склеенные в один список. **all_answers f []** должен возвращать **Some []**.

Следующие задачи связаны с реализацией сопоставления по шаблонам. Типы для этих задач будут описываться следующим образом:

```
type pattern = Wildcard | Variable of string | UnitP
| ConstP of int | TupleP of pattern list |
ConstructorP of string * pattern

type value = Const of int | Unit | Tuple of value
list | Constructor of string * value
```

Шаблон **pattern p** либо подходит к значению **value v**, либо нет. Если да, сопоставление порождает список пар **(string * value)** - связывания для этого шаблона. Правила сопоставления следующие:

- **Wildcard** подходит к любым значениям и порождает пустой список связываний.
- **Variable s** подходит к любому значению **v** и порождает список связываний из единственного элемента **(s, v)**.
- **UnitP** подходит только к **Unit** и порождает пустой список связываний.
- **ConstP 42** подходит только к **Const 42** и порождает пустой список связываний (С остальными константами аналогично).
- **TupleP ps** сопоставляется со значением вида **Tuple vs**, если списки **ps** и **vs** имеют одинаковую длину, и каждый элемент **ps** подходит соответствующему элементу **vs**. Результирующий список связываний получается склеиванием списков, порождённых вложенными шаблонами.
- **ConstructorP(s1, p)** сопоставляется с **Constructor(s2, v)** если строки **s1** и **s2** равны и **p** подходит к **v**. Итоговый список связываний - список, полученный после сопоставления вложенного шаблона.
- В других случаях сопоставление неудачно.

9. (+1) В этой задаче воспользуйтесь предоставленной функцией **g**.
    a. С помощью **g** определите функцию **count_wildcards**, которая принимает на вход **pattern** и возвращает количество шаблонов вида **wildcard**, которые он в себе содержит.
    b. С помощью функции **g** определите функцию **count_wild_and_variable_lengths**, которая принимает на вход **pattern** и возвращает сумму длин всех

имён переменных, которые в нём встречаются (имена конструкторов считать не нужно), плюс количество встречающихся в нём шаблонов вида **`Wildcard`**.

c. С помощью **`g`** определите функцию **`count_some_var`**, которая принимает пару **`string * pattern`** и возвращает количество вхождений этой строки в шаблон в виде имени переменной (имена конструкторов считать не нужно)

10. (+2) Напишите функцию **`check_pat`**, которая принимает **`pattern`** и возвращает **`true`** тогда и только тогда, когда все переменные в нём различны. Подсказка: **`List.foldBack`** и **`List.exists`** могут здесь пригодиться.
11. (+2) Напишите функцию **`match_pat`**, которая принимает кортеж **`value * pattern`** и возвращает **`(string * value) list option`**: **`None`**, если исходный шаблон не подходит, и **`Some lst`**, где **`lst`** - список порождаемых связываний, если подходит. Обратите внимание, что если шаблон подходит, но не содержит переменные (следовательно, не порождает связывания), то результат должен быть **`Some []`**. Подсказка: воспользуйтесь предыдущими решениями (например, **`all_answers`**) и полезными библиотечными функциями вроде **`List.zip`**.
12. (+1) Напишите функцию **`first_match`**, которая принимает **`value`** и **`pattern list`**, и возвращает **`(string * value) list option`**: **`None`**, если ни один шаблон из списка не подходит, или **`Some lst`**, где **`lst`** - список связываний, порождаемый первым подошедшим шаблоном. Подсказка: воспользуйтесь **`first_aswer`**.
13. Задача-бонус (+5). Напишите функцию **`typecheck_patterns`**, которая проверяет типы для списка шаблонов. Типы будут определяться следующим образом:

```
type typ =
      | Anything
      | UnitT
      | IntT
      | TupleT of typ list
      | Type of string
```

Функция **typecheck_patterns** должна иметь тип **((string * string * typ) list) * (pattern list) -> typ option**. Первый аргумент содержит элементы вида **("foo", "bar", UnitT)**, что означает "конструктор **foo**, который создаёт значение типа **bar** из значения типа **UnitT**. Например:

("foo", "bar", IntT) означает что-то вроде
type bar = ... | foo of int ...

("foo", "bar", UnitT)
type bar = ... | foo ...
(UnitT - это тип для ничего)

Считайте, что имена конструкторов (первое поле) различны. Исходя из этих предположений, выведите такой тип **t**, который могут иметь все шаблоны списка **pattern list**. Если такой тип существует, верните **Some t**, в противном случае верните **None**.

Обратите внимание, что возвращать нужно "наименее строгий" тип из возможных. Например, для шаблонов **TupleP[Variable("x"); Variable("y")]** и **TupleP[Wildcard; Wildcard]** верните **TupleT[Anything; Anything]**, несмотря на то, что оба они могли бы иметь тип **TupleT[IntT; IntT]**. Другой пример: для шаблонов **TupleP[Wildcard; Wildcard]** и **TupleP[Wildcard; TupleP[Wildcard; Wildcard]]** верните **TupleT[Anything; TupleT[Anything; Anything]]**.

В результате, в файле с решением должны быть следующие функции:

```
val only_capitals : (System.String list ->
System.String list)
  val longest_string1 : (System.String list ->
System.String)
  val longest_string2 : (System.String list ->
System.String)
  val longest_string3 : (System.String list ->
System.String)
  val longest_string4 : (System.String list ->
System.String)
```

```
  val longest_capitalized : (System.String list ->
System.String)
  val rev_string : s:string -> System.String
  val first_answer : f:('a -> 'b option) -> lst:'a
list -> 'b
  val all_answers : f:('a -> 'b list option) ->
lst:'a list -> 'b list option
  val count_wildcards : (pattern -> int)
  val count_wild_and_variable_lengths : (pattern ->
int)
  val count_some_var : x:string * p:pattern -> int
  val check_pat : pat:pattern -> bool
  val match_pattern : valu:value * pat:pattern ->
(string * value) list option
  val first_match :
    valu:value -> patlst:pattern list -> (string *
value) list option
```