Задание №5
Обработка списков и неопределенных значений. Шаблоны

1. Общая постановка задачи

На языке Standard ML опишите реализацию функции, соответствующей номеру вашего варианта.

При выполнении задания нужно соблюдать следующие ограничения:

- При описании функций НЕ ДОЛЖНО БЫТЬ ЯВНЫХ УКАЗАНИЙ ТИПОВ аргументов и результата;
- Нельзя использовать функции `#1`, `#2`, `hd`, `tl`, `valOf`, `isSome`, `List.take`, `List.drop` (вместо этого следует обходиться использованием шаблонов). Кроме того, нельзя определять и использовать аналоги этих функций;
- Нельзя использовать функции `foldr`, `foldl`, `map`, `List.filter`, `List.find`;
- Все необходимые циклические процессы должны быть реализованы посредством хвостовой рекурсии.

В файле с программой приведите несколько вызовов функции, демонстрирующих корректную работу в различных ситуациях.

Файлу с программой дайте имя `task5-NN.sml`, где `NN` — номер вашего варианта. Полученный файл загрузите на портал в качестве решения задания.

2. Предварительные замечания

Не следует выполнять никаких предварительных преобразований заданных выражений, в частности сокращений дробей.

Не следует проводить предварительных вычислений элементов последовательности, не заданных явно в задании.

Вспомогательные функции и значения должны определяться только в качестве локальных. Результат загрузки файла с решением в интерпретатор — только определение реализуемой функции.

Не следует делать предположений на счет задания, не сформулированных явно в условии. Если возникают сомнения — задайте вопрос на форуме «Язык Standard ML».

3. Пример выполнения задания

Опишите функцию `delDupl : ''a list -> * ''a list`, принимающую список элементов, и возвращающую список, полученный из исходного удалением подряд идущих дубликатов элементов. Например, `delDupl [1, 2, 2, 2, 3, 3, 4, 5] = [1, 2, 3, 4, 5]`.

РЕШЕНИЕ: Содержимое файла `task5-NN.sml`:

```
fun delDupl x =
  let
    (* Вспомогательная функция определяется с помощью шаблонов.
     * Первый аргумент функции - обрабатываемый список,
     * второй - формируемый в обратном порядке результат *)
      (* Если первый аргумент - пустой список, то результат
       * сформирован, и остается его только перевернуть с
       * помощью стандартной функции rev *)
    fun delDuplIter ([],  res) = rev res
      (* Если в обрабатываемом списке только один элемент,
       * то включаем его в результат и проводим рекурсивный
       * вызов *)
      | delDuplIter ([x], res) = delDuplIter ([], x :: res)
      (* В противном случае - в списке есть два первых элемента,
       * которые обозначаем через x1 и x2, при этом хвост списка
       * обозначаем через xs. Если x1 и x2 совпадают,
       * x1 выбрасываем. В противном случае,
       * x1 помещаем в результат *)
      | delDuplIter (x1 :: (xs as x2 :: _), res) =
          if x1 = x2 then delDuplIter (xs, res)
          else  delDuplIter (xs, x1 :: res)
  in
    delDuplIter (x, [])
  end

(* ТЕСТОВЫЕ ЗАПУСКИ *)
val test0 = delDupl []
val test1 = delDupl [12]
val test2 = delDupl [1, 2, 2, 2, 3, 3, 4, 5]
val test3 = delDupl [ 111, 111, 111, 2, 2, 2, 2, 3, 3, 3, 2, 2, 2, 1
                    , 1, 1, 2, 3, 3, 43, 4, 5, 6, 6, 4, 4, 4, 4 ]
val test4 = delDupl ["a", "b", "b", "b", "c", "c", "d", "e"]
```

Текст примера можно загрузить с портала.

4. Варианты заданий

1. Опишите функцию `alternate : int list -> int`, получающую список целых чисел и подсчитывающую сумму его элементов с переменой знака. Например, `alternate [1, 2, 3, 4] = 1 - 2 + 3 - 4 = -2`.

2. Опишите функцию `minMax : int list -> (int * int) option`, получающую список целых чисел и возвращающую `SOME (min, max)`, где `min` — минимальный, а `max` — максимальный элемент списка. Если в заданном списке нет элементов, то функция должна возвращать `NONE`.

3. Опишите функцию `cumSum : int list -> int list`, получающую список целых чисел и возвращающую список частичных сумм его элементов. Например, `cumSum [1, 4, 20] = [1, 5, 25]`.

4. Опишите функцию `greeting : string option -> string`, получающую значение `SOME name`, где `name` строка, представляющая имя. Функция должна возвращать строку вида `"Hello there, ...!"`, где многоточие заменяет имя `name`. Если функция получает `NONE`, то вместо имени в результат должно быть подставлено слово `"you"`.

5. Опишите функцию `repeat : int list * int list -> int list`, получающую два списка: список целых чисел и список неотрицательных целых чисел. Функция должна в списке-результате повторять каждый элемент первого списка то количество раз, которое указано в соответствующем элементе второго списка. Например, `repeat ([1, 2, 3], [4, 0, 3]) = [1, 1, 1, 1, 3, 3, 3]`. Если количество элементов в первом и втором списке не совпадает, то необходимо обрабатывать минимальное количество из предложенных элементов списка.

6. Опишите функцию `addAllOpt : int option list -> int option`, получающую список «опциональных» значений. Функция «складывает» все элементы списка, представляющие числа. Если в списке ни одного числа не представлено, результат — `NONE`. Например, `addAllOpt [SOME 1, NONE, SOME 3] = SOME 4`,
`addAllOpt [] = NONE`

7. Опишите функцию `zip : int list * int list -> (int * int) list`, получающую два числовых списка и возвращающую список пар, в котором каждая i-я пара содержит i-е элементы заданных списков. Если длина заданных списков отличается, то в результирующем списке должно быть столько же элементов, сколько в списке с минимальной длиной. Например, `zip ([1, 2, 3], [4, 6]) = [(1, 4), (2, 6)]`.

8 (50% бонус)**.** Опишите функцию `zipRecycle : int list * int list -> (int * int) list`, получающую два непустых числовых списка и возвращающую список пар, в котором каждая i-я пара содержит i-е элементы заданных списков. Если длина заданных списков отличается, то в результирующем списке должно быть столько же элементов, сколько в списке с максимальной длиной, причем элементы списка с минимальной длиной должны циклически повторяться. Например,

```
zipRecycle ([1, 2, 3], [1, 2, 3, 4, 5, 6, 7]) =
          [(1, 1), (2, 2), (3, 3), (1, 4), (2, 5), (3, 6), (1, 7)].
```

9 (70% бонус)**.** Опишите функцию `zipOpt : int option list * int list -> (int * int) list`, получающую два списка и возвращающую список пар, в которой i-я пара содержит на второй позиции i-й элемент второго списка, а на первом — i-е представленное число первого списка. Если в первом списке не представлено чисел, то результатом должен быть пустой список. В противном случае результат должен содержать столько же пар, сколько элементов во втором списке. Если в первом списке представлено чисел меньше, чем элементов во втором списке, то представленные числа должны циклически повторяться. Например,

```
zipOpt ([SOME 1, NONE, NONE, SOME 2, SOME 3], [1, 2, 3, 4, 5, 6, 7]) =
          [(1, 1), (2, 2), (3, 3), (1, 4), (2, 5), (3, 6), (1, 7)].
```

10. Опишите функцию `lookup : (string * int) list * string -> int option`, получающую список пар `(str, i)` и строку `str2`. Если в заданном списке найдется пара, на первой позиции которой стоит строка `str2`, то функция выдает `SOME j`, где `j` — второй элемент этой пары. В противном случае функция выдает `NONE`.

11. Опишите функцию `splitup : int list -> int list * int list`, получающую числовой список и возвращающую пару списков, где первый содержит все неотрицательные числа из заданного списка, а второй — все отрицательные. Порядок элементов в списках должен сохраниться прежним.

12. Опишите функцию `splitAt : int list * int -> int list * int list`, получающую числовой список и число. Функция должна возвращать пару списков, где первый содержит все элементы заданного списка, превышающие по величине второй аргумент, а второй — все остальные элементы. Порядок элементов в списках должен сохраниться прежним.

13. Опишите функцию `isAnySorted : int list -> boolean`, получающую числовой список и выдающую `true`, если список отсортирован по убыванию или возрастанию.

14. Опишите функцию `sortedMerge : int list * int list -> int list`, получающую два отсортированных в порядке возрастания числовых списка и возвращающую объединенный список всех элементов, отсортированный в том же порядке. Например, `sortedMerge ([1, 4, 7], [5, 8, 9]) = [1, 4, 5, 7, 8, 9]`.

15. Опишите функцию `divideAlt : 'a list -> 'a list * 'a list`, получающую числовой список и разделяющую его, чередуя элементы по двум спискам. Например,
`divideAlt [1, 2, 3, 4, 5, 6, 7] = ([1, 3, 5, 7], [2, 4, 6])`.

16. Опишите функцию `mucSum : int list -> int list`, формирующую из заданного списка список-результат следующим образом: первый элемент — сумма всех элементов, второй — сумма элементов хвоста, третий — сумма элементов от третьего до последнего, и т. д.. Например, `mucSum [3, 15, 9, 33] = [60, 57, 42, 33]`.

17 (50% бонус)**.** Опишите функцию `sameOrder : ''a list * ''a list -> bool`, принимающую два списка и возвращающую `true`, если все элементы, входящие в оба списка, расположены в этих списках в одном и том же порядке. Считаем, что в каждом списке нет повторяющихся элементов.

18. Опишите функцию `delKth : 'a list * int -> 'a list`, принимающую список `lst` и число `k`. Функция должна возвращать список, полученный из `lst` удалением каждого `k`-ого элемента. Например,
`delKth ([3, 1, 3, 2, 3, 5, 4, 4, 7, 9, 6, 6], 4) = [3, 1, 3, 3, 5, 4, 7, 9, 6]`.

19 (50% бонус)**.** Опишите функцию `maxSorted : int list -> int list`, принимающую список и извлекающую из него максимальную по длине последовательность соседних элементов, упорядоченных по возрастанию. Если таких последовательностей несколько, то необходимо вернуть первую из них. Например,
`maxSorted [3, 1, 3, 2, 3, 5, 4, 4, 7, 9, 6] = [2, 3, 5]`.

20 (30% бонус)**.** Опишите функцию `intersect : ''a list * ''a list -> * ''a list`, принимающую два списка и возвращающую список элементов, входящих в оба списка, причем упоминая каждый элемент только один раз. Например,
`intersect ([3, 3, 2, 3, 4, 4, 7, 9, 6], [1, 6, 3, 3, 5, 10, 6, 9]) = [3, 9, 6]`.