Задание №19 Бесконечные списки

1. Общая постановка задачи

Решение должно использовать определения, приведённые в файле `streams.lsp`. В файле `streams.lsp` даны функции, макросы и значения для работы с отложенными вычислениями и с потоками (с бесконечными списками).
Здесь определены:

`delay` — макрос для отложения вычисления заданного выражения;

`force` — функция для извлечения значения отложенного вычисления;

`cons-stream` — макрос для организации бесконечного списка;

`head` — функция извлечения головы бесконечного списка;

`tail` — функция получения хвоста бесконечного списка;

`TES` — константа — пустой поток (аналог `NIL` для бесконечных списков);

`empty-stream-p` — функция тестирующая поток на пустоту (аналог `null` для бесконечных списков);

Кроме того, здесь присутствуют определения двух потоков: `*ones*` — бесконечная последовательность единиц; `(integers-starting-from n)` — последовательность целых чисел, начинающаяся с `n`.

Решение задания должно быть оформлено в отдельном файле, в котором первая команда подгружает определения из файла `streams.lsp`.

Опишите функцию `drop`, принимающую в качестве параметров целое число `n` и поток (бесконечный список). Функция должна выдавать заданный поток без первых `n` элементов. В случае недостаточного количества элементов в потоке необходимо удалять то количество элементов, которое есть в наличии (то есть, возвращать пустой поток).

Опишите функцию `take`, принимающую целое число `n` и поток (бесконечный список). Функция должна выдавать список (обычный список) из первых `n` элементов потока. В случае недостаточного количества элементов в потоке необходимо возвращать список из того количества элементов, которое есть в наличии.

На языке Lisp необходимо описать реализацию бесконечной последовательности `f19`, соответствующей номеру вашего варианта задания 12.

Поток должен определяться в том же порядке, в котором он определялся при решении задания 12. Должны быть определены и использованы аналоги функций и значений, используемых при решении задания 12. Исключение из этого правила (см. пример в файле `task19-00.lsp`): начальные элементы последовательности могут быть добавлены по отдельности с использованием `cons-stream` вместо использования аналога функции для соединения конечного списка с бесконечным; порядок связывания последовательности, зависящей от параметров с именем переменной (для рекурсивного определения) может быть отличен от порядка связывания в Haskell.

Имена функций и переменных в программе должны повторять имена аналогов в решении задания 12 с оглядкой на правила именования, оговариваемые правилами оформления (см. примеры).

Поток должен быть определён как результат функции, если для его задания необходимы параметры. Если для определения потока параметров не требуется, то он должен быть сохранен в переменной.

Код программы должен быть прокомментирован. Из комментариев должно быть очевидно, как код программы на Lisp соотносится с кодом программы на Haskell. При отсутствии таких комментариев решение может не проверяется и оцениваться в 0 баллов.

Задайте набор тестовых вызовов, демонстрирующих выполнение вашего задания. Тестовые вызовы должны использовать реализованные функции `take` и `drop`.

Файлу с программой дайте имя `task19-NN.lsp`, где вместо `NN` — номер вашего варианта задания 12. Полученный файл загрузите на портал в качестве решения задания.

Содержимое файла `streams.lsp`:

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; блок для реализации отложенных вычислений
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; макрос, создающий обещание (thunk из выражения expr)
(defmacro delay (expr)
  `(cons NIL (lambda () ,expr)))

;; функция вычисления обещания
(defun force (delay)
  (if (car delay)
      (cdr delay)
      (progn (setf (car delay) T)
             (setf (cdr delay) (funcall (cdr delay))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; блок для реализации потоков
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; макрос для создания потока из головы a и хвоста str
;; результат вычисления str – поток
(defmacro cons-stream (a str)
  `(cons ,a (delay ,str)))

;; функция извлечения головы потока (вместо car-stream)
(defun head (stream)
  (car stream))

;; функция извлечения хвоста потока (вместо cdr-stream)
(defun tail (stream)
  (force (cdr stream)))

;; определение глобальной переменной – пустой поток
;; (The Empty Stream)
```

```lisp
(defvar TES 'TES)

;; функция проверки потока на пустоту
(defun empty-stream-p (stream)
  (eq stream TES))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; примеры потоков
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; глобальная переменная, содержащая поток единиц
(defvar *ones*) ; объявление переменной
(setq *ones* (cons-stream 1 *ones*))

;; функция, выдающая поток целых чисел, начинающийся с n
(defun integers-starting-from (n)
  (cons-stream n (integers-starting-from (1+ n))))
```

2. Пример выполнения задания

0 (Пример 1. Вычисление элемента по трём предыдущим (повторение задания для языка Haskell))**.**

Опишите функцию `f12 :: Integral a => a -> a -> a -> [a]` , определяющую для заданных значений a_0, a_1, a_2 последовательность, в которой каждый последующий элемент вычисляется по следующим правилам:

$$a_{n+1} = \begin{cases} a_n - a_{n-1}, & \text{если } n \bmod 2 = 0 \text{ и } a_n > 0 \\ (a_n + a_{n-1} - 2 * a_{n-2}^2) \bmod 100, & \text{если } a_n \leqslant 0 \\ (a_{n-1} + 2 * a_{n-2}) \bmod 5, & \text{если } n \bmod 2 = 1 \text{ и } a_n > 0 \end{cases}$$

Последовательность должна быть составлена из своих же частей с помощью функции `zipWith` . Первым аргументом `zipWith` должна передаваться вспомогательная функция, реализующая заданную формулу вычисления очередного элемента последовательности.

РЕШЕНИЕ:

Содержимое файла `task19-00.lsp` :

```lisp
(load "streams.lsp")
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; здесь должна быть реализация функций take и drop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; пример, соответствующий содержимому файла task12-00.hs
;; в качестве пар (кортежа двух значений) используем точечные пары

;; функция - аналог zipWith
;; получает два потока (бесконечных списка)
;; формирует поток
(defun zip-with-stream (f stream1 stream2)
  (if (or (empty-stream-p stream1) (empty-stream-p stream2))
      TES
      (cons-stream (funcall f (head stream1) (head stream2))
                   (zip-with-stream f (tail stream1) (tail stream2)))))

;; функция - аналог nextEl
;; получает две пары (точечные пары) и вычисляет результат
;; по заданной формуле
(defun next-el (pair1 pair2
                &aux
                (a0 (car pair1)) (a1 (cdr pair1))
                (a2 (car pair2)) (n (cdr pair2)))
  (cond
    ((<= a2 0) (mod (+ a2 a1 (* -2 a0 a0)) 100))
    ((= 0 (mod n 2)) (- a2 a1))
    (T (mod (+ a1 (* 2 a0)) 5))))

;; функция - аналог f12 a0 a1 a2
(defun f19 (a0 a1 a2)
  (cons-stream a0     ; вместо объединения первых трех элементов
    (cons-stream a1   ; последовательности в список, добавляем их в поток
      (cons-stream a2 ; в обратном порядке
        (let* ((seqn (f19 a0 a1 a2))    ; аналог значения seqn
```

```
              (seqn-tail (tail seqn))) ; аналог значения seqnTail
          (zip-with-stream
            'next-el
            (zip-with-stream 'cons seqn seqn-tail) ; используем cons для
            (zip-with-stream 'cons                 ; объединения в пары
                             (tail seqn-tail)      ; как аналог (,)
                             (integers-starting-from 2))))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; тестовые примеры
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
; 6-й элемент
(print (head (tail (tail (tail (tail (tail (f19 0 1 -1))))))))
; 8-й элемент
(print
  (head (tail (tail (tail (tail (tail (tail (tail (f19 0 0 1)))))))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; следующие примеры должны заработать после реализации функций drop и take
;; результат вывода должен совпадать с результатом запуска функции main,
;; определнной в task12-00.hs

; (print (take 15 (f19 0 1 -1)))
; (print (take 15 (f19 0 0 1)))
; (print (take 15 (drop 100 (f19 0 1 -1))))
; (print (take 15 (drop 100 (f19 1 -1 -1))))
; (print (take 15 (drop 1000 (f19 197 -11 -112))))
```

0 (Пример 2. Последовательность простых чисел (повторение задания для языка Haskell)).
Опишите последовательность `f12 :: [Integer]` — список всех простых чисел (начиная с 2). Простым числом называется натуральное (целое положительное) число, имеющее ровно два различных натуральных делителя — единицу и самого себя.

РЕШЕНИЕ:
Содержимое файла `task19-00-1.lsp` :

```
(load "streams.lsp")
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; здесь должна быть реализация функций take и drop

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; пример, соответствующий содержимому файла task12-00-1.hs

;; функция - аналог divides
;; так как в task12-00-1.hs функция была каррированная и это свойство
;; использовалось в решении (функция вызывается от первого аргумента),
;; то определяем её как каррированную - функцию первого аргумента,
;; возвращающую функцию от второго аргумента
(defun divides (x)
  (lambda (y)
    (= (mod y x) 0)))

;; функция - аналог iterate
(defun iterate (f x)
  (cons-stream x (iterate f (funcall f x))))

;; функция - аналог filter
;; поток stream фильтруется в соответствии с предикатом p
(defun filter-stream (p stream)
  (cond ((empty-stream-p stream) TES)
        ((funcall p (head stream))
         (cons-stream (head stream)
                      (filter-stream p (tail stream))))
        (T (filter-stream p (tail stream)))))

;; функция - аналог map
;; получает поток (бесконечный список)
;; формирует поток
(defun map-stream (f stream)
  (if (empty-stream-p stream) TES
      (cons-stream (funcall f (head stream))
                   (map-stream f (tail stream)))))
```

```lisp
;; функция, аналог композиции (в Haskell - . )
;; предполагается, что второй аргумент - функция одного параметра
(defun o (f1 f2)
  (lambda (x) (funcall f1 (funcall f2 x))))

;; функция - аналог crossout
(defun crossout (stream
                 &aux
                 (x (head stream))
                 (xs (tail stream)))
  (filter-stream (o 'not (divides x)) xs))

;; так как последовательность не зависит от параметров, определим её как
;; значение переменной f19
(defvar f19)
(setq f19
      (map-stream 'head
                  (iterate 'crossout
                           (integers-starting-from 2))))

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;;; 4. тестовые примеры
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
(print (head (tail (tail (tail (tail (tail f19)))))))  ; 6-е простое число
(print
  (head (tail (tail (tail (tail (tail (tail (tail f19)))))))))  ; 8-е число

;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; следующие примеры должны заработать после реализации функций drop и take
;; результат вывода должен совпадать с результатом запуска функции main,
;; определнной в task12-00-1.hs

; (print (take 15 f19))
; (print (take 15 (drop 100 f19)))
; (print (take 5 (drop 1000 f19)))
```

3. Необходимый минимум

Для выполнения работы потребуются сведения о следующих функциях, операциях и конструкциях:

- функции `cons`, `car`, `cdr`
- конструкцию **`if`**
- конструкции **`let`**, **`let*`**
- логические, арифметические функции, функции сравнения
- конструкцию `defun`
- конструкцию **`lambda`**
- предикат `null`
- функции `funcall`, `apply`
- функцию вывода на экран `print`
- математические функции, аналоги функций, используемых в решении задания 12

Если вы считаете, что для выполнения какого-то из заданий необходима функция/конструкция, отсутствующая в перечислении, то задайте вопрос на форуме «Язык LISP».