Дважды в одну реку

Язык: Racket

Максимальное количество баллов: (не считая бонусной задачи): 13

В этой работе нужно будет поупражняться с функциями, обрабатывающими списки, и стримами. Обратите внимание, что в некоторых задачах требуется написать обобщённые стримы, с помощью которых можно комбинировать другие стримы. Воспользуйтесь ими как кирпичиками для решения остальных задач. Если нужно, можно определять вспомогательные стримы для этих целей.

1. (+1) Напишите функцию **`sequence`**, которая принимает аргументы **`low`**, **`high`** (целые числа) и **`stride`** (положительное число). Функция **`sequence`** должна возвращать список чисел от **`low`** до **`high`** шагом **`stride`**. Например:

```
(sequence 3 11 2) -> '(3 5 7 9 11)
(sequence 3 8 3) -> '(3 6)
(sequence 3 2 1) -> '()
```

2. (+1) Напишите функцию **`string-append-map`**, которая принимает список строк **`xs`** и строку **`suffix`** и возвращает список строк, присоединив к каждой строке исходного списка строку **`suffix`** (без пробелов между строкой и суффиксом). Используйте библиотечные функции **`map`** и **`string-append`**.
3. (+1) Напишите функцию **`list-nth-mod`**, которая принимает список xs и число n. Если n отрицательное, остановите выполнение с **`(error "list-nth-mod: negative number")`**. Если список пустой, остановите выполнение с **`(error "list-nth-mod: empty list")`**. Иначе, верните i-й элемент списка, где i - остаток от деления **`n`** на длину исходного списка. Подсказка: библиотечные функции **`length`**, **`remainder`**, **`car`**, **`list-tail`** могут здесь пригодиться.
4. (+1) Напишите функцию **`stream-for-n-steps`**, которая принимает стрим **`s`** и число **`n`**, и возвращает список из первых n значений, порождённых стримом. Считайте, что n неотрицательное.
5. (+1) Напишите функцию **`stream-map`**, которая принимает стрим **`s`** и функцию **`f`**, и возвращает стрим, каждый элемент которого

является результатом применения **f** к соответствующим элементам исходного стрима.

6. (+1) Напишите функцию **combine-streams**, которая принимает стримы **s1** и **s2** и функцию **f**, и возвращает стрим, каждый элемент которого является результатом применения **f** к соответствующим элементам первого и второго стрима.
   Например:
   s1 - 1, 2, 3…
   s2 - 2, 3, 4 …
   (combine-streams s1 s2 +) - 3, 5, 7 …
7. (+1) Напишите функцию **skip-first-n**, которая принимает стрим **s** и число **n**, и возвращает стрим, возвращающий элементы исходного стрима, начинающиеся с **n+1** по счёту.
8. (+1) Напишите функцию **funny-number-stream**, которая похожа на стрим натуральных чисел за исключением того, что все числа, кратные 5 - отрицательные (то есть 1, 2, 3, 4, -5, 6, 7, 8, 9, -10 …).
   Напишите функцию **squares**, которая принимает стрим **s** и возвращает стрим, элементы которого являются квадратами элементов исходного стрима. Помните, что стрим - это thunk, который при вызове возвращает пару (результат . новый стрим).
9. (+2) Напишите функцию **taylor**, которая принимает число **x**, и возвращает стрим, элементы которого являются слагаемыми ряда Тейлора для натурального логарифма по формуле:
   $$ln(x + 1) = \sum_{n=1}^{\infty} \frac{(-1)^{n-1} x^n}{n}$$
10. (+1) Напишите функцию **cycle-lists**, которая принимает два списка **xs** и **ys** и возвращает стрим (списки могут быть разной длины, но не пустые). Элементы стрима - это пары, где первая часть - элемент из **xs**, вторая - элемент из **ys**. Например, если **xs** равен **'(1, 2, 3)**, а **ys - ("a", "b")**, то стрим будет генерировать значения **(1 . "a"), (2 . "b"), (3 . "a"), (1 . "b"), (2 . "a"), (3 . "b"), (1 . "a"), (2 . "b")** и т. д.

11. (+2) Напишите функцию **vector-assoc**, которая принимает значение **v** и вектор **vec**. Её поведение должно быть аналогично библиотечной функции **assoc**, за исключением того, что она обрабатывает вектор, а не список, позволяет элементам вектора не быть парами (в этом случае они просто пропускаются) и всегда принимает ровно два аргумента. Воспользуйтесь функциями **vector-length**, **vector-ref** и **equal?**. Верните **#f**, если ни один элемент вектора не является парой с головой, равной **v**, иначе верните первую такую пару.
12. Задача-бонус (+3) Напишите функцию **cached-assoc**, которая принимает список **xs** и число **n** и возвращает функцию **f**, которая возвращает то же самое, что вернул бы вызов **(assoc v xs)**. Реализуйте кэш на **n** последних результатов (будет эффективнее, чем обычный **assoc**, если список **xs** длинный, но только небольшое количество элементов ищется в нём часто). Кэш - это вектор длиной **n**, который создаётся во время вызова **cached-assoc**, а используется (и, возможно, модифицируется) каждый раз, когда вызывается функция **f**. **n** считайте положительным.

    Сначала кэш пустой (все элементы равны **#f**). Когда вызывается функция **f**, она сначала ищет результат в кэше. Если его там нет, она использует **assoc** и **xs**, получает результат, и если он не **#f** (нашлась подходящая пара), то результат добавляется в кэш (с помощью **vector-set!**) и затем возвращается. Кэш работает по принципу кольцевого буфера. Подсказка: храните переменную-указатель на свободный кэш-слот и меняйте её значение с помощью **set!**.