Лабораторная работа №1 (Вариант II)

Рекурсия, процедуры высшего порядка, обработка списков

Цель работы

Приобретение навыков работы с основами программирования на языке Scheme: использование рекурсии, процедур высшего порядка, списков.

Вопросы для допуска к работе

1. Процедура (функция): определение понятия, способы определения в языке Scheme.
2. Точечные пары и списки. Представление правильного списка с помощью точечных пар. Встроенные процедуры `cons`, `car`, `cdr`.
3. Назначение процедур `append`, `apply`.
4. Что такое свертка. Какие встроенные процедуры в языке Scheme обладают свойствами свертки?
5. Особенности реализации `and` и `or` в языке Scheme.

Задания

При выполнении заданий **не используйте** присваивание, циклы и обращение к элементам последовательности по индексу. Избегайте возврата логических значений из условных конструкций. Продемонстрируйте работоспособность процедур на примерах.

1. Реализуйте процедуру `(count x xs)`, подсчитывающую, сколько раз встречается элемент `x` в списке `xs`. Примеры применения процедуры:

   ```
   (count 'a '(a b c a)) ⇒ 2
   (count 'b '(a c d))   ⇒ 0
   (count 'a '())        ⇒ 0
   ```

2. Реализуйте процедуру `(delete pred? xs)`, которая “удаляет” из списка `xs` все элементы, удовлетворяющие предикату `pred?`. Примеры применения процедуры:

   ```
   (delete even? '(0 1 2 3)) ⇒ (1 3)
   (delete even? '(0 2 4 6)) ⇒ ()
   (delete even? '(1 3 5 7)) ⇒ (1 3 5 7)
   (delete even? '()) ⇒ ()
   ```

3. Реализуйте процедуру `(iterate f x n)`, которая возвращает список из `n` элементов вида $(x, f(x), f(f(x)), f(f(f(x))), \ldots)$, где `f` — процедура (функция) одного аргумента. Примеры применения процедуры:

   ```
   (iterate (lambda (x) (* 2 x)) 1 6) ⇒ (1 2 4 8 16 32)
   (iterate (lambda (x) (* 2 x)) 1 1) ⇒ (1)
   (iterate (lambda (x) (* 2 x)) 1 0) ⇒ ()
   ```

4. Реализуйте процедуру `(intersperse e xs)`, которая возвращает список, полученный путем вставки элемента `e` между элементами списка `xs`. Примеры применения

процедуры:

```
(intersperse 'x '(1 2 3 4)) ⇒ (1 x 2 x 3 x 4)
(intersperse 'x '(1 2))     ⇒ (1 x 2)
(intersperse 'x '(1))       ⇒ (1)
(intersperse 'x '())        ⇒ ()
```

5. Реализуйте предикаты `(any? pred? xs)`, который возвращает `#t`, если хотя бы один из элементов списка `xs` удовлетворяет предуикату `pred?`, и `(all? pred? xs)`, который возвращает `#t`, если все элементы списка `xs` удовлетворяет предуикату `pred?`. **Не используйте** условные конструкции, вместо них используйте особенности встроенных `and` и `or`. Примеры применения:

```
(any? odd? '(1 3 5 7)) ⇒ #t
(any? odd? '(0 1 2 3)) ⇒ #t
(any? odd? '(0 2 4 6)) ⇒ #f
(any? odd? '()) ⇒ #f

(all? odd? '(1 3 5 7)) ⇒ #t
(all? odd? '(0 1 2 3)) ⇒ #f
(all? odd? '(0 2 4 6)) ⇒ #f
(all? odd? '()) ⇒ #t ; Это - особенность, реализуйте её
```

6. Реализуйте композицию функций (процедур) одного аргумента, для чего напишите процедуру `o`, принимающую произвольное число процедур одного аргумента и возвращающую процедуру, являющуюся композицией этих процедур. Примеры применения процедуры:

```
(define (f x) (* x 2))
(define (g x) (* x 3))
(define (h x) (- x))

((o f g h) 1) ⇒ -6
((o f g) 1)   ⇒ 6
((o h) 1)     ⇒ -1
((o) 1)       ⇒ 1
```

7. Реализуйте процедуру `(find-number a b c)`, возвращающее наименьшее число в интервале `[a, b]`, которое без остатка делится на `c`, или `#f`, если такое число не может быть найдено. Примеры применения процедуры:

```
(find-number 0 5 2) ⇒ 0
(find-number 7 9 3) ⇒ 9
(find-number 3 7 9) ⇒ #f
```