Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ «Информатика и системы управления»

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

Отчет по лабораторной работе №6 по дисциплине «Функциональное и логическое программирование»

Тема Использование функционалов

Студент Зайцева А. А.

Группа ИУ7-62Б

Оценка (баллы) _____

Преподаватели Толпинская Н.Б., Строганов Ю. В.

Москва — 2022 г.

Практические задания

Используя функционалы:

1. Напишите функцию, которая уменьшает на 10 все числа из списка-аргумента этой функции.

1. Если все элементы списка – числа.

```
; используя функционал mapcar
(defun all_minus_10 (lst)
  (mapcar #'(lambda (x) (- x 10)) lst)
)

; используя функционал mapcan
(defun all_minus_10 (lst)
  (mapcan #'(lambda (x) (cons (- x 10) Nil)) lst)
)

(all_minus_10 '(0 10 -10 5.5 2/3)) => (-10 0 -20 -4.5 -28/3)
(all_minus_10 Nil) => NIL
```

2. Элементы списка – любые объекты.

а) Работа только по верхнему уровню.

```
; используя функционал mapcar.
(defun all_minus_10 (lst)
  (mapcar
    #'(lambda (x) (cond ((numberp x) (- x 10)) (x)))
    lst
  )
)

; рекурсивно
; а) с помощью рекурсии, собирающей результат на выходе
(defun all_minus_10 (lst)
  (cond
    ((null lst) Nil)
    ((numberp (car lst)) (cons (- (car lst) 10) (all_minus_10 (cdr lst)
       )))
    (t (cons (car lst) (all_minus_10 (cdr lst))))
  )
)


```

```
; б) через "хвостовую рекурсию"
(defun all_minus_10_inner (lst result)
  (cond
    ((null lst) (reverse result))
    ((numberp (car lst)) (all_minus_10_inner (cdr lst) (cons (- (car
       lst) 10) result)))
    (t (all_minus_10_inner (cdr lst) (cons (car lst) result)))
  )
)

(defun all_minus_10 (lst)
  (all_minus_10_inner lst ())
)


(all_minus_10 '(0 a "abc" (1 k) 2/3)) => (-10 A "abc" (1 K) -28/3)
```

б) Работа по всем уровням структурированного списка.

```
; вспомогательная функция, так как mapcar работает только со списками
; (не с точечными парами).
(defun all_minus_10_cons (cns)
  (cond
    ((and (numberp (car cns)) (numberp (cdr cns))) (cons (- (car cns)
       10) (- (cdr cns) 10)))
    ((numberp (car cns)) (cons (- (car cns) 10) (cdr cns)))
    ((and (numberp (cdr cns)) (atom (car cns))) (cons (car cns) (- (cdr
        cns) 10)))
    ((and (numberp (cdr cns)) (consp (car cns))) (cons (
       all_minus_10_cons (car cns)) (- (cdr cns) 10)))
    ((consp (car cns)) (cons (all_minus_10_cons (car cns)) (cdr cns)))
    (t (cons (car cns) (cdr cns))))
)
; используя функционал mapcar.
; Для определения того, является ли x точечной парой, используется проверка (atom (cdr x))
; вместо (consp x) из-за следующих результатов:
; (listp '(k . 10))=(consp '(k . 10)) => T)
; (consp '(1 (K . 10)))=(listp '(1 (K . 10))) => T
(defun all_minus_10 (lst)
  (mapcar
    #'(lambda (x)
      (cond
        ((numberp x) (- x 10))
        ((atom x) x)
        ((atom (cdr x)) (all_minus_10_cons x))
        ((listp x) (all_minus_10 x))

      ))
    lst))
```

```
28
29 ; рекурсивно
30 ; а) с помощью рекурсии, собирающей результат на выходе
31 (defun all_minus_10 (lst)
32   (cond
33     ((null lst) Nil)
34     ((numberp (car lst)) (cons (- (car lst) 10) (all_minus_10 (cdr lst)
         )))
35     ((atom (car lst)) (cons (car lst) (all_minus_10 (cdr lst))))
36     ((atom (cdr (car lst))) (cons (all_minus_10_cons (car lst)) (
         all_minus_10 (cdr lst))))
37     (t (cons (all_minus_10 (car lst)) (all_minus_10 (cdr lst))))
38   )
39 )
40
41 ; б) через "хвостовую рекурсию"
42 (defun all_minus_10_inner (lst result)
43   (cond
44     ((null lst) (reverse result))
45     ((numberp (car lst)) (all_minus_10_inner (cdr lst) (cons (- (car
         lst) 10) result)))
46     ((atom (car lst)) (all_minus_10_inner (cdr lst) (cons (car lst)
         result)))
47     ((atom (cdr (car lst))) (all_minus_10_inner (cdr lst) (cons (
         all_minus_10_cons (car lst)) result)))
48     (t (all_minus_10_inner (cdr lst) (cons (all_minus_10_inner (car lst
         ) ()) result)))
49   )
50 )
51
52 (defun all_minus_10 (lst)
53   (all_minus_10_inner lst ())
54 )
55
56
57 (all_minus_10 '(0 a "abc" (1 (k . 10)) 2/3 ((1 . f) . 10))) =>
58   (-10 A "abc" (-9 (K . 0)) -28/3 ((-9 . F) . 0))
```

2. Напишите функцию, которая умножает на заданное число-аргумент все числа из заданного списка-аргумента, когда:

1. Все элементы списка – числа.

```
; используя функционал mapcar
(defun mult_all (lst num)
  (mapcar #'(lambda (x) (* x num)) lst)
)
(mult_all '(0 10 -10 5.5 2/3) 2) => (0 20 -20 11.0 4/3)
```

2. Элементы списка – любые объекты.

```
; используя функционал mapcar.
(defun mult_all (lst num)
  (mapcar
    #'(lambda (x) (cond ((numberp x) (* x num)) (x)))
    lst
  )
)

; рекурсивно
; а) с помощью рекурсии, собирающей результат на выходе
(defun mult_all (lst num)
  (cond
    ((null lst) Nil)
    ((numberp (car lst)) (cons (* (car lst) num) (mult_all (cdr lst)
       num)))
    (t (cons (car lst) (mult_all (cdr lst) num)))
  )
)

; б) через "хвостовую рекурсию"
(defun mult_all_inner (lst result num)
  (cond
    ((null lst) (reverse result))
    ((numberp (car lst)) (mult_all_inner (cdr lst) (cons (* (car lst)
       num) result) num))
    (t (mult_all_inner (cdr lst) (cons (car lst) result) num))
  )
)

(defun mult_all (lst num)
  (mult_all_inner lst () num)
)


(mult_all '(0 a "abc" (1 k) 2/3) 2) => (0 A "abc" (1 K) 4/3)
```

3. Написать функцию, которая по своему списку-аргументу lst определяет является ли он палиндромом (то есть равны ли lst и (reverse lst))

1. Проверка на равенство исходного списка и инвертированного исходного списка.

```
(defun is_palindrome (lst)
  (equalp lst (reverse lst))
)
```

2. Проверка на равенство первой половины исходного списка и инвертированной второй половины исходного списка (если список нечетной длины, то центральный элемент не попадает ни в первый, ни во второй список).

(nthcdr n lst) выполняет для списка lst операцию cdr n раз, и возвращает результат. (floor n) усекает значения по нижней границе. (ceiling n) усекает значения по верхней границе.

```
(defun first_n (n lst)
  (cond
    ((or (null lst) (= n 0)) Nil)
    (t (cons
      (car lst)
      (first_n (- n 1) (cdr lst))
      )
    )
  )
)

(defun is_palindrome (lst)
  (let ((half_len (/ (length lst) 2)))
    (equalp
      (first_n (floor half_len) lst)
      (reverse (nthcdr (ceiling half_len) lst))
    )
  )
)
```

3. Рекурсивно: сравнить первый и последний элемент исходного списка, первый и последний элемент исходного списка без первого и последнего элемента, и так далее (если длина списка нечетная, то центральный элемент ни с чем не сравнивается).

```
(defun list_without_last (lst)
  (cond
    ((null (cdr lst)) Nil)
    (t (cons
      (car lst)
      (list_without_last (cdr lst))
      )
    )
  )
)

(defun is_palindrome (lst)
  (cond
    ((null (cdr lst)) t)
    ((eql (car lst) (car (last lst))) ;т. к. (last '(1 2))=>(2)
      (is_palindrome (list_without_last (cdr lst))))
  )
)
```

Все варианты функций проверялись на следующих тестах:

```
(is_palindrome Nil) => T
(is_palindrome '(1)) => T
(is_palindrome '(1 2 3)) => NIL
(is_palindrome '(1 2 1)) => T
(is_palindrome '(1 2 3 1)) => NIL
(is_palindrome '(1 2 2 1)) => T
```

4. Написать предикат set-equal, который возвращает t, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения

Все элементы первого множества последовательно удаляются из обоих множеств. Если исходные множества эквиваленты, то в конце получим два пустых множества.

```
(defun set-equal (set1 set2)
  (cond
    ((null set1) (null set2)) ;3 тест
    ((null set2) Nil) ;4 тест
    (t (set-equal (cdr set1) (remove (car set1) set2)))
  )
)

(set-equal '(1 2 3) '(1 2 3)) => T
(set-equal '() '()) => T
(set-equal '(1 2) '(1 2 3)) => NIL
(set-equal '(1 2) '(1)) => NIL
```

5. Написать функцию которая получает как аргумент список чисел, а возвращает список квадратов этих чисел в том же порядке.

1. Используя функционал mapcar

```
(defun squares (lst)
  (mapcar #'(lambda (x) (* x x)) lst)
)

(squares '(0 10 -10 5.5 2/3)) => (0 100 100 30.25 4/9)
```

2. Рекурсивно

```
; а) с помощью рекурсии, собирающей результат на выходе
(defun squares (lst)
  (cond
    ((null lst) Nil)
    (t (let ((x (car lst))) (cons (* x x) (squares (cdr lst)))))
  )
)


```

```
; б) через "хвостовую рекурсию"
(defun squares_inner (lst result)
  (cond
    ((null lst) (reverse result))
    (t (squares_inner (cdr lst) (let ((x (car lst))) (cons (* x x)
       result))))
  )
)

(defun squares (lst)
  (squares_inner lst ())
)
```

6. Напишите функцию, select-between, которая из списка-аргумента, содержащего только числа, выбирает только те, которые расположены между двумя указанными границами-аргументами и возвращает их в виде списка (упорядоченного по возрастанию списка чисел (+ 2 балла)).

(Границы не включительно)
Сортировка:

```
; блочная (карманная, корзинная) сортировка
(defun my_sort (lst)
  (cond ((null lst) Nil)
    (t (nconc
      (my_sort (remove-if-not (lambda (x) (> (car lst) x)) (cdr lst)))
      (remove-if-not (lambda (x) (= (car lst) x)) lst)
      (my_sort (remove-if-not (lambda (x) (< (car lst) x)) (cdr lst)))
      )
    )
  )
)

; сортировка выбором

; в эту функцию не должен попадать пустой список
; (лямбда-функция принимает строго 2 аргумента)
(defun list_max (lst)
  (reduce
    #'(lambda (a b) (cond ((> a b) a) (b)))
    lst
  )
)

```

```
24
25 (defun my_sort_inner (lst res)
26   (cond
27     ((null lst) res)
28     (t (let*
29          (
30            (cur_max (list_max lst))
31            (lst_rest (remove cur_max lst :count 1))
32          )
33          (my_sort_inner lst_rest (cons cur_max res))
34        )
35     )
36   )
37 )
38
39 (defun my_sort (lst)
40   (my_sort_inner lst ())
41 )
```

1. Используя функционал remove-if-not, сортировка после выделения элементов, которые расположены между двумя указанными границами-аргументами.

```
1 (defun select-between-inner (lst a b)
2   (remove-if-not #'(lambda (x) (< a x b)) lst)
3 )
4
5 (defun select-between (lst a b)
6   (cond
7     ((< a b) (select-between-inner lst a b))
8     (t (select-between-inner lst b a))
9   )
10 )
11 (defun select-between-sorted (lst a b)
12   (my_sort (select-between lst a b))
13 )
```

2. Рекурсивно. Сначала исходный список сортируется, а затем обрезается: от первого элемента, большего нижней границы, до последнего элемента, меньшего верхней границы

```
1 ; в функциях select-greater и select-lower предполагается, что переданный список
2 ; отсортирован по возрастанию
3 (defun select-greater (lst a)
4   (cond
5     ((null lst) Nil)
6     ((< a (car lst)) lst)
7     (t (select-greater (cdr lst) a))
8   )
9 )
```

```
10
11 (defun select-lower (lst b)
12   (cond
13     ((null lst) Nil)
14     ((<= b (car lst)) Nil)
15     (t (cons (car lst) (select-lower (cdr lst) b)))
16   )
17 )
18
19 (defun select-between-sorted (lst a b)
20   (select-greater (select-lower (my_sort lst) b) a)
21 )
22
23 (select-between-sorted '(0 3 7 6 5 4 1) 1 6) => (3 4 5)
```

7. Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов. (Напомним, что A x B это множество всевозможных пар (a b), где a принадлежит A, принадлежит B.)

1. С помощью функционалов mapcar и mapcan

```
1 (defun decart (lstx lsty)
2   (mapcan
3     #'(lambda (x) (mapcar
4           #'(lambda (y) (cons x (cons y Nil)))
5           lsty
6           )
7       )
8     lstx
9   )
10 )
11
12 (decart '(a b) '(1 2)) => ((a 1) (a 2) (b 1) (b 2))
```

2. Рекурсивно

```
; по принципу move-to с лекции
; составляет список из каждого элемента lst и cons-ячейки cns
(defun decart_row (lst cns res)
  (cond
    ((null lst) res)
    (t (decart_row (cdr lst) cns (cons (cons (car lst) cns) res)))
  )
)

; по принципу move-to с лекции
; (nconc вместо cons, lsty неизменен, lstx уменьшается в размере)
(defun decart_inner (lstx lsty res)
  (cond
    ((null lstx) res)
    (t (decart_inner
        (cdr lstx)
        lsty
        (nconc
          (decart_row lsty (cons (car lstx) Nil) ())
          res
        )
      )
    )
  )
)

(defun decart (lstx lsty)
  (decart_inner lstx lsty ())
)

(decart '(a b) '(1 2)) => ((2 B) (1 B) (2 A) (1 A))
```

8. Почему так реализовано reduce, в чем причина?

Функционал reduce выполняет следующее преобразование исходного списка L=>(e1 e2 ... en) с использованием начального значения A и бинарной операции-функции F: (reduce F L A) = (F(...(F(F A e1) e2))...en). Можно сказать, что значение A логически добавляется в начало списка L.

- Если список содержит ровно один элемент и начальное значение не задано, то этот элемент возвращается, а функция не вызывается.
- Если список пуст и задано начальное значение, то возвращается начальное значение, а функция не вызывается.
- Если список пуст и начальное значение не задано, то функция вызывается без аргументов, и reduce возвращает то, что вернет функция. Это единственный случай, когда функция вызывается с другим количеством аргументов, кроме двух.

В требованиях сказано, что функция должна принимать в качестве аргументов два элемента последовательности или результаты объединения этих элементов. Функция также должна быть способна не принимать никаких аргументов.

```
1 (reduce #'+ ()) => 0
2 ; в задании указано, что (reduce #'+ 0) => 0, но
3 (reduce #'+ 0) => Value of 0 in (REDUCE #'+ 0) is 0, not a SEQUENCE.
4 ; возможно, имелось в виду следующее:
5 (reduce #'+ () :initial-value 0) => 0
6 ; или
7 (reduce #'+ '(0)) => 0
```

(reduce #'+ () :initial-value 0): список пуст, начальное значение указано, следовательно, функция + не вызывается, и reduce возвращает начальное значение.

(reduce #'+ '(0)): список состоит из одного элемента, начальное значение не указано, следовательно, функция + не вызывается, и reduce возвращает единственный элемент списка.

(reduce #'+ ()): список пуст, начальное значение не указано, следовательно, функция + вызывается без аргументов, и reduce возвращает то, что вернет функция +. (+) => 0, поэтому и (reduce #'+ ()) => 0.

Примечание: (reduce #'* ()) => по той же причине, но для функций - и / такой вызов осуществить невозможно, так как они не могут принимать ноль аргументов.

9. Пусть list-of-list список, состоящий из списков. Написать функцию, которая вычисляет сумму длин всех элементов list-of-list, т.е. например для аргумента ((1 2) (3 4)) -> 4

```
; с использованием функционалов mapcar и reduce
(defun sum_of_legths (ll)
  (reduce #'+ (mapcar #'length ll))
)

; рекурсивно
(defun sum_of_legths_inner (ll sum)
  (cond
    ((null ll) sum)
    (t (sum_of_legths_inner (cdr ll) (+ sum (length (car ll)))))
  )
)

(defun sum_of_legths (ll)
  (sum_of_legths_inner ll 0)
)

(sum_of_legths '()) => 0
(sum_of_legths '(())) => 0
(sum_of_legths '((1 2))) => 2
(sum_of_legths '((1 2) (3 4))) => 4
(sum_of_legths '((1 2) (3) (4 5))) => 5
```

Литература