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»

Лабораторная работа №8
Использование функционалов

Студент: Луговой Д.М.
Группа: ИУ7-61Б
Преподаватель: Толпинская Н.Б.

Москва, 2020 г.

Цель работы: приобрести навыки работы в управляющими структурами Lisp.

Задачи работы: изучить работу функций с произвольным количеством аргументов, функций, разрушающих и не разрушающих структуру исходных аргументов.

Лабораторная работа 5

Задание 1

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

Листинг 1: 1-я реализация функции проверки на палиндром

```
(defun check_pal(lst)
  (cond
    (
      (null lst)
    )
    (
      (and
        (equal
          (car lst)
          (mapcon #'(lambda (el)
              (cond
                (
                  (null (cdr el))
                  (car el)
                )
              )
            ) lst
          )
        )
        (check_pal
          (mapcon #'(lambda (el)
              (cond
                (
                  (cdr el)
                  (cons (car el) nil)
                )
              )
            ) (cdr lst)
          )
        )
      )
    )
```

```
33    )
34 )
```

lst - входной список.

Условием выхода из рекурсии является достижение середины списка(аргумент - nil) - возвращается t.

Осуществляется проверка на равенство первого и последнего элементов списка, если они равны, то осуществляется рекурсивный вызов текущей функции для списка-аргумента без первого и последнего аргументов. Иначе возвращается nil.

Листинг 2: 2-я реализация функции проверки на палиндром

```
1  (defun check_pal (lst)
2    (cond
3      (
4        (null lst)
5      )
6      (
7        (reduce
8          #'(lambda (x y) (and x y))
9          (mapcar #'equal lst (reverse lst))
10       )
11     )
12   )
13 )
```

lst - входной список.

Осуществляется полный перебор входного списка и его перевернутой с помощью reverse копии. С помощью mapcar сравниваются их элементы, а затем с помощью reduce проверяется, есть ли несовпадающие элементы.

Примеры работы:

```
1  > (check_pal '(a))
2  T
3  > (check_pal nil)
4  T
5  > (check_pal '(a b))
6  NIL
7  > (check_pal '(a b a))
8  T
9  > (check_pal '(a b b a))
10 T
11 > (check_pal '((a b) c d c (a b)))
12 T
```

Задание 4

Напишите функцию swap-first-last, которая переставляет в списке аргументе первый и последний элементы.

Листинг 3: Функция перестановки первого и последнего элементов

```
(defun swap-first-last(lst)
  (cond
    (
      (null (cdr lst))
      lst
    )
    (
      (nconc
        (
          mapcon #'(lambda (el)
            (cond
              (
                (null (cdr el))
                el
              )
            )
          ) lst
        )
        (
          mapcon #'(lambda (el)
            (cond
              (
                (cdr el)
                (cons (car el) nil)
              )
            )
          ) (cdr lst)
        )
        (cons (car lst) nil)
      )
    )
  )
)
```

lst - входной список.

С помощью первого mapcon получается список из последнего элемента списка аргумента, с помощью второй mapcon получается список-аргумент без первого и последнего элементов, и создается список из первого элемента списка-аргумента. Затем эти списки объединяются с помощью nconc.

Примеры работы:

```
> (swap-first-last '(1 2 3 4))
(4 2 3 1)
```

```
> (swap-first-last nil)
NIL
> (swap-first-last '(1))
(1)
> (swap-first-last '((1 2) 3 4 (5)))
((5) 3 4 (1 2))
```

Задание 6

Напишите две функции, swap-to-left и swap-to-right, которые производят круговую перестановку в списке-аргументе влево и вправо, соответственно на k позиций.

Листинг 4: Функция круговой перестановки элементов списка влево

```
(defun swap-to-left (lst k)
  (cond
    (
      (<= k 0)
      lst
    )
    (
      (swap-to-left
        (nconc
            (cdr lst)
            (cons (car lst) nil)
        )
        (- k 1)
      )
    )
  )
)
```

lst - входной список, **k** - количество позиций, на которое необходимо выполнить перестановку.

Условием выхода из рекурсии является окончание перестановки элементов (второй аргумент - 0) - возвращается первый аргумент.

С помощью nconc объединяется хвост списка-аргумента со списком, указатель на голову которого указывает на голову списка-аргумента, а указатель на хвост - на nil. Затем осуществляется рекурсивный вызов текущей функции для созданного списка и второго аргумента, уменьшенного на 1.

Листинг 5: Функция круговой перестановки элементов списка вправо

```
(defun swap-to-right (lst k)
  (cond
    (
      (<= k 0)
      lst
```

```
    )
    (
      (swap-to-right
        (nconc
          (mapcon #'(lambda (el)
              (cond
                (
                  (null (cdr el))
                  el
                )
              )
            ) lst
          )
          (mapcon #'(lambda (el)
              (cond
                (
                  (cdr el)
                  (cons (car el) nil)
                )
              )
            ) lst
          )
        )
        (- k 1)
      )
    )
  )
)
```

lst - входной список, **k** - количество позиций, на которое необходимо выполнить перестановку.

Условием выхода из рекурсии является окончание перестановки элементов (второй аргумент - 0) - возвращается первый аргумент.

С помощью первого mapcon осуществляется получение списка, указатель на голову которого указывает на последний элемент списка-аргумента, а указатель на хвост - на nil. С помощью второго mapcon осуществляется получение списка-аргумента без последнего элемента. Затем с помощью nconc происходит объединение этих двух списков. Далее осуществляется рекурсивный вызов текущей функции для созданного списка и второго аргумента, уменьшенного на 1.

Примеры работы:

```
> (swap-to-right nil 3)
NIL
> (swap-to-right '(1) 3)
(1)
> (swap-to-right '(1 2 3 4 5) 1)
```

```
(5 1 2 3 4)
> (swap-to-right '(1 2 3 4 5) 3)
(3 4 5 1 2)
> (swap-to-right '(1 2 3 4 5) 5)
(1 2 3 4 5)
> (swap-to-right '((1) 2 3 (4 5)) 1)
((4 5) (1) 2 3)
> (swap-to-left '(1 2 3 4 5) 1)
(2 3 4 5 1)
> (swap-to-left '(1 2 3 4 5) 3)
(4 5 1 2 3)
> (swap-to-left '(1 2 3 4 5) 5)
(1 2 3 4 5)
> (swap-to-left '((1) 2 3 (4 5)) 1)
(2 3 (4 5) (1))
```

Задание 7

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

а) все элеметны списка - числа,

б) элементы списка - любые объекты.

Листинг 6: Функция умножения для списка из чисел

```
(defun mult_num (lst k)
  (mapcar
    #'(lambda (el) (* el k))
    lst
  )
)
```

lst - входной список, **k** - число, на которое выполняется умножение.

С помощью mapcar осуществляется формирование списка, состоящего из элементов списка-аргумента, умноженных на число-аргумент.

Листинг 7: Функция умножения для списка из любых объектов

```
(defun mult_all (lst k)
  (mapcar #'(lambda (el)
      (cond
        (
          (numberp el)
          (* el k)
        )
        (
          (atom el)
```

```
10            el
11          )
12          (
13            (mult_all el k)
14          )
15        )
16      ) lst
17    )
18  )
```

lst - входной список, **k** - число, на которое выполняется умножение.

С помощью mapcar существляется проход по всему списку и проверка типа каждого элемента:

- если он является числом, то производится умножение на число-аргумент функции и возвращается результат;
- если он является атомом, то он возвращается;
- если он является списком, то к нему рекурсивно применяется текущая функция.

Примеры работы:

```
1 > (mult_num nil 1)
2 NIL
3 > (mult_num '(1) 3)
4 (3)
5 > (mult_num '(1 2 3 4 5) 3)
6 (3 6 9 12 15)
7 > (mult_all nil 2)
8 NIL
9 > (mult_all '(1) 2)
10 (2)
11 > (mult_all '(1 2 3) 3)
12 (3 6 9)
13 > (mult_all '((1 2) 3 (4 (5))) 3)
14 ((3 6) 9 (12 (15)))
15 > (mult_all '(1 2 (a)) 3)
16 (3 6 (A))
```

Задание 8

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.

Листинг 8: Функция выборки из списка по границам

```
(defun select-between (lst a b)
  (remove-if  #'(lambda (x)
      (and
        (or (< x a)(> x b))
        (or (> x a)(< x b))
      )
    )lst
  )
)
```

lst - входной список, **a**, **b** - границы выбора.

С помощью функционала remove-if осуществляется проход по всему списку-аргументу и удаление тех элементов списка, которые не входят в заданные границы-аргументы.

Примеры работы:

```
> (select-between nil 1 2)
NIL
> (select-between '(1) 0 2)
(1)
> (select-between '(1 2 3) 4 5)
NIL
> (select-between '(1 2 3 4 5) 2 4)
(2 3 4)
> (select-between '(1 2 3 4 5) 4 2)
(2 3 4)
```