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»

Отчёт
к лабораторной работе № 7
По курсу: «Функциональное и логическое программирование»
Тема: «Работа интерпретатора Lisp.»

Студент Прохорова Л. А.

Группа ИУ7-63Б

Оценка (баллы) _____

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

Москва.
2021 г.

Цель работы: приобрести навыки работы с управляющими структурами Lisp.
Задачи работы: изучить работу функций с произвольным количеством аргументов, функций разрушающих и неразрушающих структуру исходных аргументов.

Задание 1 Написать функцию, которая по своему списку-аргументу lst определяет является ли он палиндромом (то есть равны ли lst и (reverse lst)).

С использованием функционала:

```
(defun my_reverse(lst)
	(reduce
		#'(lambda (res tmp)
			(cons tmp res)
		) lst :initial-value nil
	)
)

(defun is_pol(lst)
	(equal lst (my_reverse lst))
)
```

Тестирование

```
 (my_reverse '(1 2 3))->(3 2 1)
 (my_reverse '(1 (2 3) 4))->(4 (2 3) 1)
```

(is_pol '(1 2 1)) -> T
(is_pol '(1 2 2)) -> Nil

С использованием рекурсии:

```
(defun my_rev_rec(lst rlst)
	(cond ((null lst) rlst)
		(t (my_rev_rec (cdr lst) (cons (car lst) rlst)))
	)
)

(defun is_pol_rec (lst) (equal lst(my_rev_rec lst nil)))
```

Тестирование
(my_rev_rec '(1 2 3) nil)->(3 2 1)
(my_rev_rec '(1 (4 5) 4) nil)->(4 (4 5) 1)
(is_pol_rec '(1 3 4))-> NIL
(is_pol_rec '(1 2 1))->T

Задание 2.
Написать предикат set-equal, который возвращает t, если два его множества-аргумента содержат одни и те же элементы, порядок которых не имеет значения.

Использование функционалов

```
(defun is_in_set(el search_set)
	(reduce #'(lambda (a b) (or a b))
		(mapcar #'(lambda (x) (equal el x)) search_set)
	)
)

(defun is_subset(seta setb)
		(reduce #'(lambda (a b) (and a b))
			(mapcar #'(lambda (x) (is_in_set x setb))seta)
		)
)

(defun is_equal (seta setb)
	(and (is_subset seta setb) (is_subset setb seta))
)
```

Тестирование

(is_in_set 1 '(1 2 3))->T
(is_in_set 5 '(1 2 3))->NIL

(is_subset '(1 3 4) '(1 3 4))->T
(is_subset '(1 4 5 2) '(1 2))->NIL

(is_equal '(1 3 4) '(4 3 1))->T
(is_equal '(1 3 4 5) '(1 3 4))->NIL
(is_equal '(1 3 4) '(1 3 4 5))->NIL

С использованием стандартных функций

```
(defun is_equal (seta setb)
	(and (subsetp seta setb) (subsetp setb seta))
)
```

С использованием рекурсии

```
(defun is_in_set_r (el search_set)
	(cond ((null search_set) nil)
		((equal el (car search_set)) T)
```

```
            (t (is_in_set_r el (cdr search_set)))
    )
)

(defun is_subset_r (seta setb)
    (cond ((null seta) t)
            ((is_in_set_r (car seta) setb) (is_subset_r (cdr seta) setb))
            (t nil)
    )
)

(defun is_equal (seta setb)
    (and (is_subset_r seta setb) (is_subset_r setb seta))
)
```

Тестирование
(is_in_set_r 1 '(3 2 1))->T
(is_in_set_r 1 '(4 3 2))->NIL

(is_subset_r '(2 3) '(1 2 5))->NIL
(is_subset_r '(1 2 3) '(5 4 3 2 1))->T
(is_subset_r '() '(1 2 3))->T

(is_equal '(1 2 3) '(3 2 1))->T
(is_equal '(1 2 3 4) '(1 2 3))->NIL
(is_equal '(1 2) '(1 2 3))->NIL
(is_equal '(1 2) '(4 5 6))->NIL

Задание 3.
Напишите необходимые функции, которые обрабатывают таблицу и точечных пар:(страна, столица), и возвращают по стране - столицу, по столице - страну.

С использованием функционалов

```
(defun find_by_func(val lst)
    (find-if (lambda (x) (not (null x)))
        (mapcar #'(lambda (pair)
            (cond ((equal (car pair) val) (cdr pair))
            ((equal (cdr pair) val) (car pair))
            ))lst)
    )
)
```

Тестирование
(find_by_func `moscow `((russia . moscow) (italy . rim))) ->RUSSIA
(find_by_func 'russia '((russia . moskow)(italy . rim)))->MOSKOW
(find_by_func 'ukrain '((russia . moskow)(italy . rim)))->NIL

С использованием рекурсии
```
(defun find_by_func_r (val lst)
	(cond ((null lst) nil)
		((equal (caar lst) val) (cdar lst))
		((equal (cdar lst) val) (caar lst))
		(t (find_by_func_r val (cdr lst)))
	)
)
```

Тестирование
(find_by_func_r `moscow `((russia . moscow) (italy . rim))) ->RUSSIA
(find_by_func_r 'russia '((russia . moskow)(italy . rim)))->MOSKOW
(find_by_func_r 'ukrain '((russia . moskow)(italy . rim)))->NIL

Задание 4
Напишите функцию swap-first-last, которая переставляет в списке-аргументе первый и последний элементы.
```
(defun my_reverse(lst)
	(reduce
		#'(lambda (res tmp)
			(cons tmp res)
		) lst :initial-value nil
	)
)


(defun swap-first-last (list)
   (reduce
      #'(lambda (result tmp)
         (cond ((equal (length result) (- (length list) 1))
               (my_reverse (cons  (car list) result)))
              ( t (cons tmp result ))
         )
      ) (cdr list)  :initial-value (last list)
   )
)
```

Тестирование
(swap-first-last '(1 2 3 4 5))->(5 2 3 4 1)
(swap-first-last '(1))->(1)

(swap-first-last ()) ->NIL

Задание 5. Напишите функцию swap-two-ellement, которая переставляет в списке- аргументе два указанных своими порядковыми номерами элемента в этом списке.

```
;возвращает n-ый хвост
(defun my_nthcdr (n list)
   (reduce
      #'(lambda (result tmp)
         (if (equal (length result) (- (length list) n))
            result
            (cdr result)
         )
      ) list :initial-value list
   )
)

;возвращает n
(defun my_nth (n list)
   (car (my_nthcdr n list))
)

(defun swap-two-ellement (left right lst)
   (reduce
      #'(lambda (result tmp)
         (cond
            (
               (equal (length result) left)
               (append result (list (my_nth right lst)))
            )

            (
               (equal (length result) right)
               (append result (list (my_nth left lst)))
            )

            (
               t
               (append result (list tmp))
            )
         )
      ) lst :initial-value nil
   )
)
```

Тестирование
(my_nthcdr 3 '(1 2 3 4 5))->(4 5)
(my_nthcdr 8 '(1 2 3 4 5))->NIL
(my_nth 3 '(1 2 3 4 5))->4
(my_nth 8 '(1 2 3 4 5))->NIL
(sw'(ap-two-ellement 0 3 '(1 2 3 4))->(4 2 3 1)
(swap-two-ellement 0 8 '(1 2 3 4)) ->(NIL 2 3 4)
(swap-two-ellement 3 0 '(1 2 3 4)) ->(4 2 3 1)

Задание 6. Напишите две функции, swap-to-left и swap-to-right, которые производят круговую перестановку в списке-аргументе влево и вправо, соответственно.

```
(defun swap-to-left (lst)
   (reduce
      #'(lambda (tmp result)
         (cons  tmp result))
    (cdr lst) :initial-value (list (car lst)) :from-end t
   )
)

(defun my_reverse(lst)
   (reduce
      #'(lambda (res tmp)
         (cons tmp res)
      ) lst :initial-value nil
   )
)
(defun swap-to-right (lst)
   (reduce
      #'(lambda (result tmp)
         (cond ((equal (length result) (length list)) (my_reverse result))
            ( t (cons tmp result))
         )
      ) lst :initial-value (last lst)
   )
)
```

Тестирование

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

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

Ответы на теоретические вопросы:

1.Способы определения функций

Определение именованной функции – функция defun. Принимает на вход ровно 3 аргумента: имя функции (символьный атом), список формальных параметров и тело функции (s-выражение)

Общий вид:

(defun <имя функции> (<формальные параметры>) (<тело функции>))

Пример:

(defun sum2(a b) (+ a b))

Вызвать функцию, определенную defun можно по имени.

Определение неименованной функции – с помощью лямбда-выражения.

Общий вид лямбда-выражения:

(lambda (<список формальных параметров>) (<тело функции>))

(lambda (<список формальных параметров>) (<тело функции>) <фактические параметры>)

Пример:

(lambda (x y) (+ (* x x) (* y y)))

(lambda (x y) (+ (* x x) (* y y)) 2 3) => 13

Функцию, определенную с помощью lambda-выражения можно вызвать с помощью funcall или apply:

(funcall <lambda-выражение> <фактические параметры>)

(apply <lambda-выражение> (<список фактических параметров>))

Функцию, определенную с помощью лямбда-выражения можно передать в качестве параметра в функционалы, это *будет эффективнее*, чем передавать именованную функцию по имени – так как нет необходимости переходить по указателю при каждом вызове функции.Пример:

(reduce (lambda (x y) (cons y x)) ‘(1 2 3) :initial-value Nil) => (3 2 1)

2. Варианты и методы модификации элементов списка.

Лисп позволяет применять функцию к каждому из элементов списка (или списков) с помощью отображающих функционалов.

Функционал mapcar принимает на вход имя функции или lambda-выражение и переменное количество списков-аргументов.

Если передан только один список-аргумент: функция, переданная в параметры mapcar, применяется последовательно к каждому из значений по car-указателям списковых ячеек. Из вычисленных значений формируется список с помощью list.

Общий вид:

(mapcar #’func ‘(x1 x2 … xn)) -> (list (func x1) (func x2)… (func x3))

Пример:

(mapcar (lambda (x) (* x 2)) ‘(1 2 3)) => (2 4 6)

Если передано несколько списков-аргументов: функция, переданная в аргументы mapcar, должна иметь столько же формальных параметров, сколько было передано списков-аргументов. Функция последовательно применяется к первым элементам всех списков-аргументов, затем ко вторым и т.д. Если списки-аргументы имеют разную длину, mapcar вычисляет элементы результирующего списка, пока не закончатся элементы самого короткого из списков-аргументов.

Общий вид:

(mapcar #’func lst1 lst2 … lstn)

Пример:

(mapcar (lambda (x y z) (+ x y z)) ‘(1 2 3) ‘(2 3 4) ‘(3 4 5)) => (6 9 12)

Функционал mapcan работает аналогичным образом, однако результирующий список формируется не с помощью list, а с помощью nconc. Сравнение работы:

* (mapcar (lambda (x y z) (list x y z)) ‘(1 2 3) ‘(3 4 5) ‘(5 6 7))

((1 3 5) (2 4 6) (3 5 7))

* (mapcan (lambda (x y z) (list x y z)) ‘(1 2 3) ‘(3 4 5) ‘(5 6 7))

(1 3 5 2 4 6 3 5 7)

Функционал maplist принимает имя функции или лямбда-выражение и ровно один список-аргумент. Применяет переданную функцию ко всему списку-аргументу, а затем - последовательно к каждому последующему хвосту, переходя по cdr-указателям. Формирует результирующий список из вычисленных значений с помощью cons.

Общий вид:

(maplist #’func lst)

Пример:

(maplist (lambda (lst_i) (apply ‘+ lst_i)) ‘(1 2 3)) => (6 5 3)

Функционал mapcon работает аналогично, но формирует результирующий список с помощью nconc. Сравнение работы:

* (mapcon 'list '(1 2 3))

((1 2 3) (2 3) (3))

* (maplist 'list '(1 2 3))

(((1 2 3)) ((2 3)) ((3)))