предприятие (организация)

УДК _____
№ госрегистрации _____
Инв. №_____

УТВЕРЖДАЮ

_____
головной исполнитель НИР

_____ _____

«_____» _____ 2020 г.

ОТЧЁТ
О НАУЧНО-ИССЛЕДОВАТЕЛЬСКОЙ РАБОТЕ

_____
наименование и шифр НИР

_____
(промежуточный)
шифр — _____

2020

ВВЕДЕНИЕ

Целью работы является приобретение навыков использования функционалов и рекурсии.

Задачи работы: изучить работу и методы использования отображающих функционалов: mapcar, maplist, reduce и др., изучить способы организации хвостовой рекурсии, сравнить эффективность.

1 Теоретические сведения

— Способы организации повторных вычислений в Lisp: функционалы, рекурсия.

— Различные способы использования функционалов: функционалы-предикаты; функционалы, использующие предикаты в качестве функционального объекта.

— Что такое рекурсия? Способы организации рекурсивных функци: рекурсия — это ссылка на определяемый объект во время его определения; способы организации: хвостовая, по нескольким параметрам, дополняемая, множественная.

— Способы повышения эффективности реализации рекурсии: применять хвостовую организацию рекурсии.

2 Практическая часть

2.1 Задание №1

Условие:

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

В литинге 2.1 приведен текст решения дааной задачи на языке Common Lisp.

Листинг 2.1 — Задание №1

```
;;; fun
(defun set-equal (l1 l2)
  (and (= (length l1) (length l2))
       (reduce #'(lambda (val1 val2) (and val1 val2))
               (mapcar #'(lambda (e1)
                           (reduce #'(lambda (val1 val2) (or val1 val2))
                                   (mapcar #'(lambda (e2)
                                               (equal e1 e2))
                                           l2)))
                       l1))))

;;; rec
(defun help1 (e1 l2)
  (cond ((null l2) nil)
        (t (or (equal e1 (car l2))
               (help1 e1 (cdr l2))))))

(defun help2 (l1 l2)
  (cond ((null l1) t)
        (t (and (help1 (car l1) l2)
                (help2 (cdr l1) l2)))))

(defun set-equal (l1 l2)
  (and (= (length l1) (length l2))
       (help2 l1 l2)))

;;; test
(set-equal '(1 2 3) '(1 3 2)) ; T
(set-equal '(1 2 3) '(1 5 2 3)) ; NIL
(set-equal '(1 5 2 3) '(1 2 3)) ; NIL
```

Способ с использованием функционалов состоит в том, что необходимо составить список-маску, в котором каждый элемент соотвествует по порядку каждому элементу из первого списка-аргумента функции. Если во втором списке есть хотя бы один элемент, равный i-му элементу первого списка, то i-ый элемент маски равен **t**, иначе - **nil**. Если в списке-маске есть хотя бы один **nil**, то весь результат равен **nil**, иначе - **t**.

Рекурсивный способ решения данной задачи так же заключается в полном переборе: каждый элемент из первого списка сравнивается с каждым из второго, если нет ни одного совпадения, то ответ - **nil**, иначе - **t**.

В обоих случаях присутствует начальная проверка на совпадение длин списков-множеств, которая для равных должна совпадать.

2.2 Задание №2

Условие:

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

В литинге 2.2 приведен текст решения дааной задачи на языке Common Lisp.

Листинг 2.2 — Задание №2

```
1  ;;; fun
2  (defun table_create (countries capitals)
3    (mapcar #'cons countries capitals))
4
5  (defun get_capital (tbl country)
6    (reduce #'(lambda (val1 val2) (or val1 val2))
7            (mapcar #'(lambda (row)
8                        (cond ((equal (car row) country) (cdr row))
9                              (t nil)))
10                   tbl)))
11
12 (defun get_country (tbl capital)
13   (reduce #'(lambda (val1 val2) (or val1 val2))
14           (mapcar #'(lambda (row)
```

```
15                      (cond ((equal (cdr row) capital) (car row))
16                            (t nil)))
17                  tbl)))
18
19 ;;; rec
20 (defun table_create (countries capitals)
21   (cond ((cdr countries)
22          (cons (cons (car countries) (car capitals))
23                (table_create (cdr countries) (cdr capitals))))
24         (t
25          (list (cons (car countries) (car capitals))))))
26
27 (defun get_capital (tbl country)
28   (cond ((null tbl) nil)
29         ((equal (caar tbl) country) (cdar tbl))
30         (t (get_capital (cdr tbl) country))))
31
32 (defun get_country (tbl capital)
33   (cond ((null tbl) nil)
34         ((equal (cdar tbl) capital) (caar tbl))
35         (t (get_country (cdr tbl) capital))))
36
37 ;;; test
38 (setq countries '(Russia France Germany))
39 (setq capitals '(Moscow Paris Berlin))
40 (setq table (table_create countries capitals))
41
42 (get_capital table 'Russia) ; MOSCOW
43 (get_capital table 'Germany) ; BERLIN
44 (get_capital table 'France) ; PARIS
45 (get_capital table 'USA) ; NIL
46
47 (get_country table 'Berlin) ; GERMANY
48 (get_country table 'Moscow) ; RUSSIA
49 (get_country table 'Paris) ; FRANCE
50 (get_country table 'Washington) ; NIL
```

2.3 Задание №3

Условие:

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

а) все элементы списка - числа;

б) элементы списка – любые объекты.

В литинге 2.3 приведен текст решения дааной задачи на языке Common Lisp.

Листинг 2.3 — Задание №3

```
;;; fun
(defun mul1 (lst num)
  (mapcar #'(lambda (item)
              (* num item))
          lst))

(defun mul2 (lst num)
  (mapcar #'(lambda (item)
              (cond ((numberp item) (* num item))
                    (t item)))
          lst))

;;; rec
(defun mul1 (lst num)
  (cond ((null lst) nil)
        (t (cons (* (car lst) num) (mul1 (cdr lst) num)))))

(defun mul2 (lst num)
  (cond ((null lst) nil)
        (t (cons (cond ((numberp (car lst)) (* (car lst) num))
                       (t (car lst)))
                 (mul2 (cdr lst) num)))))

;;; test
(setq nums '(4 21 5 43 31 8))
(setq vals '(4 nil 5 43 (5 7) 8))

(identity nums) ; (4 21 5 43 31 8)
(mul1 nums 4) ; (16 84 20 172 124 32)

(identity vals) ; (4 NIL 5 43 (5 7)
(mul2 vals 4) ; (16 NIL 20 172 (5 7)
```

2.4 Задание №4

Условие:

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

В литинге 2.4 приведен текст решения дааной задачи на языке Common Lisp.

Листинг 2.4 — Задание №4

```
1  ;;; fun
2  (defun sub10 (lst)
3    (mapcar #'(lambda (val)
4                (cond ((numberp val) (- val 10))
5                      (t val)))
6            lst))
7
8  ;;; rec
9  (defun sub10 (lst)
10   (cond ((null lst) nil)
11         (t (cons (cond ((numberp (car lst)) (- (car lst) 10))
12                        (t (car lst)))
13                  (sub10 (cdr lst))))))
14
15 ;;; test
16 (setq lst1 '(-2 4 1 2 4 nil 5 8 12 (555 666) 9 -99 10 7))
17 (sub10 lst1) ; (-12 -6 -9 -8 -6 NIL -5 -2 2 (555 666) -1 -109 0 -3)
```

2.5 Задание №5

Условие:

Написать функцию, которая возвращает первый аргумент списка-аргумента, который сам является непустым списком.

В литинге 2.5 приведен текст решения дааной задачи на языке Common Lisp.

Листинг 2.5 — Задание №5

```
;;; fun
(defun find-no-empty (lst)
  (reduce #'(lambda (val1 val2) (or val1 val2))
          (mapcar #'(lambda (item)
                      (cond ((listp item) item)
                            (t nil)))
                  lst)))

;;; rec
(defun find-no-empty (lst)
  (cond ((null lst) nil)
        ((and (not (null (car lst)))
              (listp (car lst))) (car lst))
        (t (find-no-empty (cdr lst)))))

;;; test
(find-no-empty '(1 23 nil -6 Word (1 2 3) () 66 (4 5 6))) ; (1 2 3)
(find-no-empty '((3 4))) ; (3 4)
(find-no-empty '(1 2)) ; nil
(find-no-empty  '()) ; nil
```

2.6 Задание №6

Условие:

Написать функцию, которая выбирает из заданного списка только те числа, которые больше 1 и меньше 10.

В литинге 2.6 приведен текст решения дааной задачи на языке Common Lisp.

Листинг 2.6 — Задание №6

```
;;; fun
(defun filter (predicate lst)
  (reverse (reduce #'(lambda (state item)
                       (cond ((funcall predicate item) (cons item state))
                             (t state)))
                   (cons nil lst))))

;;; rec
(defun filter (predicate lst)
  (cond ((null lst) nil)
        (t (cond ((funcall predicate (car lst))
```

```
12                  (cons (car lst) (filter predicate (cdr lst))))
13                 (t (filter predicate (cdr lst)))))))
14
15 ;;; common
16 (defun interval-filter (lst lo hi)
17   (filter #'(lambda (item)
18               (cond ((numberp item) (or (and (> item lo) (< item hi))
19                                         (and (> item hi) (< item lo))))
20                     (t nil)))
21           lst))
22
23 ;;; test
24 (setq lst1 '(-2 4 1 2 4 nil 5 8 12 (555 666) 9 -99 10 7))
25
26 (interval-filter lst1 1 10) ; (4 2 4 5 8 9 7)
27 (interval-filter lst1 10 1) ; (4 2 4 5 8 9 7)
28 (interval-filter lst1 100 1000) ; NIL
29 (interval-filter lst1 1 1) ; NIL
```

2.7 Задание №7

Условие:

Написать функцию, вычисляющую декартово произведение двух своих списков-аргументов.

В литинге 2.7 приведен текст решения дааной задачи на языке Common Lisp.

Листинг 2.7 — Задание №7

```
1  ;;; fun
2  (defun cart-product (l1 l2)
3    (mapcan #'(lambda (e1)
4                (mapcar #'(lambda (e2) (list e1 e2))
5                        l2))
6            l1))
7
8  ;;; rec
9  (defun help1 (e1 l2)
10   (cond ((null l2) nil)
11         (t (cons (list e1 (car l2))
12                  (help1 e1 (cdr l2))))))
13
```

```
14 (defun cart-product (l1 l2)
15   (cond ((null l1) nil)
16         (t (nconc (help1 (car l1) l2)
17                   (cart-product (cdr l1) l2)))))
18
19 ;;; test
20 (cart-product nil '(3 4)) ; NIL
21 (cart-product '(1 2) nil) ; NIL
22 (cart-product '(buy sell) '(car cycle))
23 ; ((BUY CAR) (BUY CYCLE) (SELL CAR) (SELL CYCLE))
```

2.8 Задание №8

Условие:

Почему так реализовано reduce, в чем причина?
$(reduce\#*+0) -> 1$
$(reduce\#*+()) -> 0$

Этот текст не компилируется. Возможно, вместо него подразумевался текст, приведенный в листинге 2.8.

Листинг 2.8 — Задание №8

```
1 (reduce #'+ nil) ; 0
2 (reduce #'* nil) ; 1
```

В данном случае, результат первой строчки совпадает с результатом вычисления (+), а второй - с результатом вычисления (*), что очевидно, так как список аргументов в обоих случаях пуст (равен **nil**).

Такое поведение функции reduce приведёт к ошибке при использовании в качестве функционального объекта функции, не вычисляющейся при пустом списке аргументов.

Тем не менее, функция reduce имеет особенность: если используемый список аргументов состоит только из одного элемента, то в результате вычисления функции reduce будет получено значение именно этого элемента. Данный нюанс можно объяснить тем, что от функции reduce ожидается, что при использовании функции от двух аргументов в качестве функционально-

го объекта и списка параметров, содержащего один или более элемент, она должна вычисляться без ошибок времени выполнения.