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 №4 по дисциплине "Функциональное и логическое программирование"

Тема Использование управляющих структур, работа со списками

Студент Егорова П.А.

Группа ИУ7-64Б

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

Москва — 2023 г.

Задание 1

Постановка задачи

Чем принципиально отличаются функции `cons`, `list`, `append`?
Пусть `(setf lst1 '(a b c)) (setf lst2 '(c d))`
Каковы результаты следующих выражений?

```
(cons lst1 lst2) -> ((A B C) C D)
(list lst1 lst2) -> ((A B C) (C D))
(append lst1 lst2) -> (A B C C D)
```

Задание №2

Каковы результаты вычисления следующих выражений, и почему?

```
(reverse '(a b c)) -> (C B A)
(reverse ()) -> NIL
(reverse '(a b (c (d)))) -> ((C (D)) B A)
(reverse '((a b c))) -> ((A B C))
(reverse '(a)) -> (A)
(last '(a b c)) -> (C)
(last '(a b (c))) -> ((C))
(last '(a)) -> (A)
(last ()) -> NIL
(last '((a b c))) -> ((A B C))
```

Задание №3

Написать, по крайней мере, два варианта функции, которая возвращает последний элемент своего списка-аргумента.

Решение

Листинг 1: Решение задания №3

```
(defun last_el (lst)
  (if (cdr lst)
    (last_el (cdr lst))
    (car lst))
)

(defun last_el (lst)
  (car(reverse lst))
)
```

Задание №4

Написать, по крайней мере, два варианта функции, которая возвращает свой список-аргумент без последнего элемента.

Решение

Листинг 2: Решение задания №4

```
(defun without_last (lst)
(if (cdr lst)
    (cons (car lst) (without_last (cdr lst)))
))

(defun without_last (lst)
(reverse (cdr(reverse lst)))
)
```

Задание №5

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

Решение

Листинг 3: Решение задания №5

```
  (defun swap-first-last (lst)
  (setf f (car lst))
  (setf nl (get_change_last (cdr lst)))
  (cons l nl)
  )

  (defun get_change_last (lst)
  (if (cdr lst)
  (cons (car lst) (get_change_last (cdr lst)))
  ((lambda () (setf l (car lst)) (list f) ))
  ))
```

Задание №6

Написать простой вариант игры в кости, в котором бросаются две правильные кости. Если сумма выпавших очков равна 7 или 11 – выигрыш, если выпало $(1, 1)$ или $(6, 6)$ — игрок право снова бросить кости, во всех остальных случаях ход переходит ко второму игроку, но запоминается сумма выпавших очков. Если второй игрок не выигрывает абсолютно, то

выигрывает тот игрок, у которого больше очков. Результат игры и значения выпавших костей выводить на экран с помощью функции `print`.

Решение

Листинг 4: Решение задания №6

```
(defun get_points () (list (random 7) (random 7)))

(defun is_check_pair (pair check_pair)
(and (not (atom pair)) (equal (car pair) (car check_pair)) (equal (cadr pair
    ) (cadr check_pair))
))

(defun sum_pair (pair)
(and (not (atom pair)) (+ (car pair) (cadr pair)))
)

(defun check_absolute_win(pair)
(and (not (atom pair))
(or
    (equal (sum_pair pair) 7)
    (equal (sum_pair pair) 11)
)
))

(defun can_rerun (pair)
    (or (is_check_pair pair '(1 1)) (is_check_pair pair '(6 6)))
)

(defun logic (points)
(if (check_absolute_win points)
    'won
    (if (can_rerun points)
    ((lambda ()
        (princ "Do you want rerun?[y if yes, any if no]")
        (terpri)
        (if (equal (read) 'y)
            (setf p1 (get_points))
        )
    ))
    )
))

(defun winner (p1 p2)
(if (> (sum_pair p1) (sum_pair p2))
    (princ "First player won!")
    (princ "Second player won!")
))

```

```
(defun game ()
        (setf p1 (get_points))
        (princ "Game started!")
        (terpri)
        (princ "First player's turn. Points: ")
        (princ p1)
        (terpri)
        (setf res1 (logic p1))
        (if (equal res1 'won)
            (princ "First player won!")
            ((lambda ()
                (if (not (equal res1 nil))
                    ((lambda ()
                    (setf p1 res1)
                    (princ "First player's points: ")
                    (princ p1)
                    (terpri)
                    ))
                 )
                (setf p2 (get_points))
                (princ "Second player's turn. Points: ")
                (princ p2)
                (terpri)
                (setf res2 (logic p2))
                (if (equal res2 'won)
                    (princ "Second player won!")
                    ((lambda ()
                        (if (not (equal res2 nil))
                            ((lambda ()
                            (setf p2 res2)
                            (princ "Second player's points: ")
                            (princ p2)
                            (terpri)
                            ))
                        )
                        (winner p1 p2)
                    ))
                )
            ))
        )
)
```

Задание №7

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

Решение

Листинг 5: Решение задания №7

```
(defun move (lst res)
(cond ((null lst) res)
(t (move (cdr lst) (cons (car lst) res) ))
) )

(defun my_reverse (lst)
(move lst ()))

(defun is_eq (lst revlst)
(cond ((and (null lst) (null revlst)))
( (eq (car lst) (car revlst)) (is_eq (cdr lst) (cdr revlst)) )
))

(defun is_poly (lst)
(is_eq lst (my_reverse lst)) )
```

Задание №8

Напишите свои необходимые функции, которые обрабатывают таблицу из 4-х точечных пар: (страна . столица), и возвращают по стране - столицу, а по столице — страну.

Решение

Листинг 6: Решение задания №8

```
(defun get_capital (lst country)
(reduce #'(lambda (x y)
(if (null x)
(if (eql (car y) country) (cdr y))
 x)) lst :initial-value Nil))

(defun get_country (lst capital)
(reduce #'(lambda (x y)
(if (null x)
(if (eql (cdr y) capital) (car y))
 x)) lst :initial-value Nil))
```

Задание №9

Напишите функцию, которая умножает на заданное число-аргумент первый числовой элемент списка из заданного 3-х элементного списка- аргумента, когда а) все элементы списка — числа, 6) элементы списка – любые объекты.

Решение

Листинг 7: Решение задания №8

```
(defun mult_only_numbers (num lst)
(cons (* num (car lst)) (cdr lst)))

(defun mul_first_num (num lst)
(setf f (car lst))
(setf s (cadr lst))
(setf th (caddr lst))
(cond
((numberp f) (cons (* num f) (cdr lst) ))
((numberp s) (list f (* num s) th ))
((numberp th) (list f s (* num th)))
(T lst)))
```

Контрольные вопросы

Вопрос 1. Синтаксическая форма и хранение программы в памяти.

Ответ. В Lisp формы представления программы и обрабатываемых ею данных одинаковы – они представлены в виде S-выражений. Программы могут обрабатывать и преобразовывать другие программы или сами себя. В памяти программа представляется в виде бинарных узлов, так как она состоит из S-выражений.

Вопрос 2. Трактовка элементов списка.

Ответ. Если отсутствует блокировка вычислений, то первый элемент списка трактуется как имя функции, а остальные элементы – как аргументы функции.

Вопрос 3. Порядок реализации программы.

Ответ. Работа программы циклична: сначала программа ожидает ввода S-выражения, затем передает полученное S-выражение интерпретатору – функции eval, а в конце, после отработки функции eval, выводит последний полученный результат.

Вопрос 4. Способы определения функций

Ответ. Существует два способа определений функций:

- через `defun`;
- через `lambda`.

Пример `defun`:

```
(defun func-name (args-list) function-body)
(defun get-cube(y) (* y y y))
(get-cube y)
```

Пример `lambda`:

```
(lambda (args-list) function-body)
((lambda (x) (* x x)) 2)
```

Вопрос 5. Работа со списками

Ответ. Функции, реализующие опреации со списками, делятся на две группы:

1. не разрушающие структуру функции; данные функции не меняют переданный им объект-аргумент, а создают копию, с которой в дальнейшем производят необходимые преобразования; к таким функциям относятся: append, reverse, last, nth, nthcdr, length, remove, subst и др.

2. структуроразрушающие функции; данные функции меняют сам объект-аргумент, из-за чего теряется возможность работать с исходным списком; чаще всего имя структуроразрушающих функций начинается с префикса -n: nreverse, nconc, nsubst и др.

Обычно в Lisp существуют функции-дубли, которые реализуют одно и то же преобразование, но по разному (с сохранением структуры и без): append/nconc, reverse/nreverse и т.д.