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 курсу «Функциональное и Логическое программирование»

на тему: «Использование управляющих структур, работа со списками»

Студент ИУ7-63Б _____ Лагутин Д. В.

(Группа) (Подпись, дата) (Фамилия И. О.)

Преподаватель _____ Толпинская Н. Б.

(Подпись, дата) (Фамилия И. О.)

Москва, 2023 г.

Теоретические вопросы

1. Синтаксическая форма и хранение программы в памяти

В Lisp формы представления программы и обрабатываемых ею данных одинаковы — S-выражение. Программы могут обрабатывать и преобразовывать другие программы или самих себя. В память программа представляется в виде списковых ячеек, так как она состоит из S-выражений.

2. Трактовка элементов списка

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

3. Порядок реализации программы

Программа выполняет следующий цикл операций:

1) ожидание ввода S-выражения;
2) обработка S-выражения интерпретатором — функция `eval`;
3) вывод последнего результата на экран.

4. Способы определения функции

Функция может быть определена двумя способами. С помощью λ-выражения `(lambda (`λ`-list) f)`, где λ-list — список формальных аргументов, а `f` - тело функции, или макро-определения `(defun name `λ`-выражение)`, где name — имя определяемой функции.

5. Работа со списками

Функции, реализующие операции со списками, делятся на две группы.

— 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.

— 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 `-п`: `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.

1 Задание

Чем принципиально отличаются функции `cons`, `list`, `append`?

`(cons object1 object2)`

Возвращает новую ячейку, `car` которой — `object1`, а `cdr` — `object2`.

`(list &rest objects)`

Возвращает новый список, состоящий из объектов `objects`.

`(append &rest prolists)`

Возвращает список с элементами из всех списков `prolists`. Последний аргумент, который может быть любого типа, не копируется.

Каковы результаты вычисления следующих выражений?

```
1  (setf lst1 '(a b c))
2  (setf lst2 '(d e))
3
4  (cons lst1 lst2)
5  ;; ((A B C) D E)
6
7  (list lst1 lst2)
8  ;; ((A B C) (D E))
9
10 (append lst1 lst2)
11 ;; (A B C D E)
```

2 Задание

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))
```

3 Задание

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

```
(defun my-last-1 (lst)
  (car (reverse lst)))

(defun my-last-2 (lst)
  (cond ((not lst) nil)
        ((not (cdr lst)) (car lst))
        ((my-last-2 (cdr lst)))))
```

4 Задание

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

```
(defun no-last-1 (lst)
  (cond ((not lst) nil)
        ((not (cdr lst)) nil)
        ((cons (car lst) (no-last-1 (cdr lst))))))

(defun no-last-2 (lst)
  (reverse (cdr (reverse lst))))
```

5 Задание

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

```
(defun reverse-set-val (lst val)
  (rplaca (nreverse lst) val))

(defun swap-first-last-1 (lst)
  (let ((last-v (car (last lst))))
    (reverse-set-val (reverse-set-val lst (car lst)) last-v)))

(defun set-last (lst val)
  (cond ((not lst) nil)
        ((not (cdr lst)) (let ((tmp (car lst))) (setf (car lst) val) tmp))
        ((set-last (cdr lst) val))))

(defun swap-first-last-2 (lst)
  (rplaca lst (set-last lst (car lst))))
```

6 Задание

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 имеет 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`.

```
1  (defun get-random-pair () (list (+ (random 6) 1) (+ (random 6) 1)))
2
3  (defun turn ()
4    (let* ((pair (get-random-pair)) (sum (+ (car pair) (cadr pair))))
5      (print "Выпавшие очки:")
6      (print pair)
7      (cond ((or (= 1 (first pair) (second pair))
8                 (= 6 (first pair) (second pair)))
9             (print "Перебросить? [y/any]: ")
10            (if (eql 'y (progn (finish-output) (read t))) (turn) sum))
11           (sum))))
12
13 (defun choose-winner (turn-1 turn-2)
14   (cond ((or (= 7 turn-1) (= 11 turn-1)) 1)
15         ((or (= 7 turn-2) (= 11 turn-2)) 2)
16         ((> turn-1 turn-2) 1)
17         ((< turn-1 turn-2) 2)
18         (0)))
19
20 (defun game ()
21   (print "ИГРА В КОСТИ")
22   (print "------------------------")
23   (let ((win (choose-winner
24                (progn (print "Игрок 1") (turn))
25                (progn (print "------------------------")
26                       (print "Игрок 2") (turn)))))
27     (print "------------------------")
28     (cond ((zerop win) (print "Ничья") nil)
29           ((= 1 win) (print "Победитель - игрок 1") nil)
30           ((= 2 win) (print "Победитель - игрок 2") nil))))
```

7 Задание

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

```
1 (defun palindrome (lst) (equal lst (reverse lst)))
```

8 Задание

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

```
1  (defun get-capital (table country)
2    (cond ((not table) nil)
3          ((equal (caar table) country) (cdar table))
4          ((get-capital (cdr table) country))))
5
6  (defun get-country (table capital)
7    (cond ((not table) nil)
8          ((equal (cdar table) capital) (caar table))
9          ((get-country (cdr table) capital))))
10
11 (defun set-capital-inner (table country capital)
12   (cond ((equal (caar table) country) (rplacd (car table) capital))
13         ((not (cdr table)) (cadr (rplacd table (list (cons country
     capital)))))
14         ((set-capital-inner (cdr table) country capital))))
15
16 (defun set-capital (table-name country capital)
17   (let ((table (eval table-name)))
18     (cond ((not table) (car (setf (symbol-value table-name) (list (cons
     country capital)))))
19           ((set-capital-inner table country capital)))))
20
21 (setf table nil)
22 (set-capital 'table Нидерланды' Амстердам')
23 (set-capital 'table Андорра' АндорралаВелья'--)
24 (set-capital 'table Греция' Афины')
25 (set-capital 'table Сербия' Белград')
26 (set-capital 'table Германия' Берлин')
27 (set-capital 'table Швейцария' Берн')
28 (set-capital 'table Словакия' Братислава')
```

9 Задание

Напишите функцию, которая умножает на заданное число-аргумент первый числовой элемент списка из заданного 3-х элементного списка-аргумента, когда

1) все элементы списка — числа,

2) элементы списка – любые объекты.

```
1 (defun mult-first-num-inner (lst val)
2   (cond ((not lst) nil)
3         ((numberp (car lst)) (rplaca lst (* (car lst) val)))
4         ((mult-first-num (cdr lst) val))))
5
6 (defun mult-first-num (lst val)
7   (mult-first-num-inner lst val)
8   lst)
```