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 Завойских Е.В.

Группа ИУ7-63Б

Оценка (баллы) _____

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

Москва — 2023 г.

1 Теоретические вопросы

1.1 Синтаксическая форма и хранение программы в памяти

Синтаксически программа оформляется в виде S-выражения, которое может быть структурированным. Наличие скобок является признаком структуры.

По определению:

- S-выражение ::= <атом> | <точечная пара>
- Атомы:
  - символы (идентификаторы) — синтаксически — набор литер (букв и цифр), начинающихся с буквы;
  - специальные символы — T, Nil (используются для обозначения логических констант);
  - самоопределимые атомы — натуральные числа, дробные числа, вещественные числа, строки — последовательность символов, заключенных в двойные апострофы (например, “abc”);
- Точечная пара ::= (<атом> . <атом>) | (<атом> . <точечная пара>) | (<точечная пара> . <атом>) | (<точечная пара> . <точечная пара>);
- Список ::= <пустой список> | <непустой список>, где

  <пустой список> ::= () | Nil,

  <непустой список> ::= (<первый элемент> . <хвост>),

  <первый элемент> ::= <S-выражение>,

  <хвост> ::= <список>.

Атомы представляются в памяти пятью указателями: name, value, function, property, package; точечная пара — списковой ячейкой (бинарным узлом), хранящей два указателя: car-указатель (на голову) и cdr-указатель (на хвост).

1.2 Трактовка элементов списка

По умолчанию список является формой (вычислимым выражением), в которой первый элемент трактуется как имя фунции, остальные элементы — как ее аргументы. Чтобы была возможность отличить программу от данных создана функция quote и ее сокращенное обозначение — апостроф '. Функция quote блокирует вычисление своего аргумента и возращает его текстовую запись.

1.3 Порядок реализации программы

Лисп-программа включает в себя:

- определения новых функций на базе встроенных функций и других функций, определённых в этой программе;
- вызовы этих новых функций для конкретных значений их аргументов.

Лисп-программа представляет собой вызов функции на верхнем уровне и синтаксически оформляется в виде S-выражения. Выполнение программы заключается в вычислении значений функций для конкретных значений аргументов. Вычисление функции происходит с помощью функции eval. Функция eval принимает в качестве аргумента S-выражение и вычисляет его. Если вычислять значение выражения не нужно, используют функцию quote.

1.4 Способы определения функции

- с использованием λ-нотации (функции без имени)

  λ-выражение: (lambda λ-список тело_функции), где λ-список — формальные параметры функции.

  Вызов такой функции осуществляется следующим способом: (λ-выражение фактические параметры).

  Вычисление функций без имени может быть выполнено с использованием функционала apply: (apply λ-выражение список_фактических_параметров); или с использованием функционала funcall: (funcall λ-выражение фактические_параметры).

- с использованием макро-определения defun:

  (defun имя_функции λ-выражение),

  или в облегченной форме:

  (defun имя_функции $(x_1, x_2, ..., x_k)$ тело_функции), где $(x_1, x_2, ..., x_k)$ — список аргументов.

  В качестве имени функции выступает символьный атом. Вызов именованной функции осуществляется следующим образом: (имя_функции фактические параметры).

1.5 Работа со списками

В Lisp существует множество функций для работы со списками:

- (cons arg1 arg2) — создает списковую ячейку. Если arg2 — список, то функция вернет список, иначе — точечную пару.
- (list arg1 arg2 arg3 ...) — создает список из значений своих аргументов.
- (append lst1 lst2 lst3 ...) — объединяет списки (работает с копиями списковых ячеек, не копируется только последний список-аргумент). nconc —альтернатива (конкатенация самих списков, а не их копий).
- (reverse lst) — переворачивает список в обратную сторону по верхнему уровню. nreverse — альтернатива (работает с самим списком, не создает копию).
- (last lst) — возвращает последнюю списковую ячейку.
- (nth n lst) — возвращает n-ый элемент списка.
- (nthcdr n lst) — возвращает n-ый хвост списка.
- (length lst) — возвращает длину списка по верхнему уровню.
- (remove elem lst) — удаляет списковую ячейку, используя функцию eql.
- (rplaca lst elem) — меняет первый элемент списка на elem.
- (rplacd lst arg) — меняет хвост списка на arg.

- (member elem lst) — возвращает список, начиная с найденного элемента elem (или Nil).
- (union lst1 lst2) — объединение множеств.
- (intersection lst1 lst2) — пересечение множеств.
- (set-difference lst1 lst2) — дополнение lst1 до lst2.
- (assoc elem table) — возвращает точечную пару, в которой ключ равен elem.
- (rassoc elem table) — возвращает точечную пару, в которой значение равно elem.

2 Практические задания

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

cons принимает 2 аргумента, создает списковую ячейку и ставит указатели на 2 аргумента, таким образом объединяя их в точечную пару.

list, составляющая список из значений своих аргументов, создает столько списковых ячеек, сколько аргументов ей было передано. Эта функция относится к особым, поскольку у неё может быть произвольное число аргументов.

append принимает произвольное количество аргументов-списков и соединяет элементы верхнего уровня всех списков в один список. Действие append иногда называют конкатенацией списков. Функцией создаются копии всех списков, кроме последнего. В результате должен быть построен новый список. Если последний переданный список будет модифицирован, то итоговый список будет также изменен.

Основные отличия: list и append принимают произвольное количество аргументов, cons — фиксированное (два); cons создает одну списковую ячейку (в зависимости от второго аргумента может получится список или точечная пара), list и append — список; cons и list создают новые списковые ячейки (все), а append имеет общие списковые ячейки с последним списком.

```
1 (setf lst1 '( a b c))
2 (setf lst2 '( d e))
3
4 (cons lst1 lst2) -> ((a b c) d e)
5 (list lst1 lst2) -> ((a b c) (d e))
6 (append lst1 lst2) -> (a b c d e)
```

2.2 Каковы результаты вычисления следующих выражений, и почему?

reverse переворачивает свой список-аргумент, т.е. меняет порядок его элементов верхнего уровня на противоположный.

last возвращает последнюю cons-ячейку в списке. Если вызывается с целочисленным аргументом n, возвращает n ячеек (по умолчанию n = 1). Если n больше или равно количеству cons-ячеек в списке, результатом будет исходный список.

```
(reverse '(a b c)) -> (c b a)
(reverse ()) -> ()
(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 ()) -> ()
(last '((a b c))) -> ((a b c))
```

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

```
(defun f1 (lst)
    (car (last lst)) )

(defun f2 (lst)
    (car (reverse lst)) )
```

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

```
(defun f1 (lst)
    (reverse (cdr (reverse lst))) )

(defun f2 (lst)
    (if (cdr lst)
        (cons (car lst) (f2 (cdr lst)) ) ) )

(defun f3 (lst) (butlast lst))
```

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

```
(defun without-last (lst)
    (reverse (cdr (reverse lst))) )

(defun swap-first-last (lst)
    (append (last lst) (without-last (cdr lst)) (list (car lst)) ) )
```

2.6 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 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.

```

(defun get_points () (+ (random 6) 1))

(defun abs_win (sum) (or (= sum 7) (= sum 11)) )

(defun roll_of_dice (player)
    (let ((ps1 (get_points)) (ps2 (get_points)))
          (print (list 'Player player 'got ps1 ps2))
```

```
          (cond
            ((abs_win (+ ps1 ps2)) 0)
            ((and (= ps1 ps2) (or (= ps1 1) (= ps1 6))) (
               roll_of_dice player))
            (T (+ ps1 ps2)) ) ) )

(defun game_result (res1 res2)
    (if (= 0 res2)
        (print "Player 2 won absolutely")
        (cond
            ((< res1 res2) (print "Player 2 won"))
            ((> res1 res2) (print "Player 1 won"))
            (T (print "Draw")) ) ) )

(defun play ()
    (let ((res1 (roll_of_dice 1)))
         (if (= 0 res1)
             (print "Player 1 won absolutely")
             (game_result res1 (roll_of_dice 2)) ) ) )
```

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

```
; v1
(defun f1 (lst)
    (equal lst (reverse lst)) )

; v2
(defun without-last (lst)
    (reverse (cdr (reverse lst))) )

(defun f2 (lst)
    (if (> (length lst) 1)
        (and (eql (car lst) (car (reverse lst)))
             (f2 (cdr (without-last lst))) )
        (and T) ) )
```

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

```
;
; v1
(defun get-capital (table country)
    (cdr (assoc country table)) )
;v2
(defun get-capital (table country)
    (cond
        ((eql (caar table) country) (cdar table))
        ((eql (caadr table) country) (cdadr table))
        ((eql (caaddr table) country) (cdaddr table))
        ((eql (caadddr table) country) (cdadddr table)) ) )

;
; v1
(defun get-country (table capital)
    (car (rassoc capital table)) )
; v2
(defun get-country (table capital)
    (cond
        ((eql (cdar table) capital) (caar table))
        ((eql (cdadr table) capital) (caadr table))
        ((eql (cdaddr table) capital) (caaddr table))
        ((eql (cdadddr table) capital) (caadddr table)) ) )
```

2.9 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, б) элементы списка – любые объекты.

```
1  ; a
2  (defun f1 (lst num)
3      (cons (* num (car lst)) (cdr lst) ) )
4
5  ; b
6  (defun f2 (lst num)
7      (cond
8          ((numberp (car lst))  (cons (* num (car lst)) (cdr lst)) )
9          ((numberp (cadr lst)) (list (car lst) (* num (cadr lst)) (
               caddr lst)) )
10         ((numberp (caddr lst)) (list (car lst) (cadr lst) (* num (
               caddr lst))) )
11         (T lst) ) )
```