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 по лабораторной работе №4 по курсу «Функциональное и логическое программирование»

Тема Использование управляющих структур, работа со списками

Студент Фам Минь Хиеу

Группа ИУ7-62Б

Оценка (баллы) _____

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

Москва — 2024 г.

Теоретические вопросы

1. Синтаксическая форма и хранение программы в памяти

Программа на Lisp представляет собой вызов функции на верхнем уровне. Все операции над данными оформляются и записываются как функции, которые имеют значение, даже если их основное предназначение – осуществление некоторого побочного эффекта. Программа является ничем иным, как набором запрограммированных функций.

Синтаксически программа оформляется в виде S-выражения (обычно – списка – частного случая точечной пары), которое очень часто 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), а любая непустая структура – списковой ячейкой (бинарным узлом), хранящей два указателя: на голову (первый элемент) и хвост – все остальное.

2. Трактовка элементов списка

По определению списка, приведенному выше: если список непустой, то он представляет из себя точечную пару из <первого элемента> и <хвоста>, где <первый элемент> – это <S-выражение>, а <хвост> – это <список>.

Список можно вычислить, если он представляет собой обращение к функции, или функциональный вызов: (f e1 e2 ... en), где f – символьный атом, имя вызываемой функции; e1, e2, ..., en – аргументы этой функции; n - число аргументов функции.

В случае n=0 имеем вызов функции без аргументов: (f). Обычно e1, e2, ..., en являются вычислимыми выражениями и вычисляются последовательно слева направо.

Таким образом, если в процессе работы лисп-интерпретатора требуется вычислить некоторый список, то первым элементом этого списка должно быть имя функции. Если это не так, лисп-интерпретатор сообщает об ошибке и прерывает вычисление текущего выражения программы.

3. Порядок реализации программы

Типичная лисп-программа включает:

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

Как отмечалось выше, программа на Lisp представляет собой вызов функции на верхнем уровне и синтаксически оформляется в виде S-выражения. Вычисление программы реализует лисп-интерпретатор, который считывает очередную входящую в программу форму, вычисляет её (анализирует функцией eval) и выводит полученный результат (S-выражение).

Eval выполняет двойное вычисление своего аргумента. Эта функция является обычной, и первое вычисление аргумента выполняет так же, как и любая обычная функция. Полученное при этом выражение вычисляется ещё раз. Такое двойное вычисление может понадобиться либо для снятия блокировки вычислений (установленной функцией quote), либо же для вычисления сформированного в ходе первого вычисления нового функционального вызова.

4. Способы определения функции

- С помощью lambda. После ключевого слова указывается лямбда-список и тело функции.

```
(lambda (x y) (+ x y))
```

  Для применения используются лямбда-выражения.

```
((lambda (x y) (+ x y)) 1 2)
```

- С помощью defun. Используется для неоднократного применения функции (в том числе рекурсивного вызова).

```
(defun sum (x y) (+ x y))
(sum 1 2)
```

Практические задания

Задание 1

Чем принципиально отличаются функции cons, list, append?

- cons является базовой функцией. list и append реализованы через cons. cons является чистой функцией и принимает 2 параметра. Она создаёт списочную ячейку, в которой car указывает на первый элемент, а cdr на второй.
- list является формой, так как принимает произвольное количество аргументов. Возвращает список из аргументов.
- append является формой, так как принимает произвольное количество аргументов. Возвращает конкатенацию аргументов. Она возвращает точечную пару, car указывает на конкатенацию всех переданных аргументов, кроме последнего, а cdr на последний аргумент.

Пусть (setf lst1 '(a b c))
(setf lst2 '(d e)).

Каковы результаты вычисления следующих выражений?

```
(setf lst1 '( a b c))
(setf lst2 '( d e))
(cons lst1 lst2)     ; -> ((a b c) d e)
(list lst1 lst2)     ; -> ((a b c) (d e))
(append lst1 lst2)   ; -> (a b c d e)
```

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.

```
(defun last1 (lst) (car (last lst)))
(defun last2 (lst) (car (reverse lst)))
```

Задание 4

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

```
(defun remove_last1 (lst) (reverse (cdr (reverse lst))))
(defun remove_last2 (lst) (reverse (last (reverse lst) (- (length lst) 1))))
```

Задание 5

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

```
(defun swap-first-last (lst)
    (cons
        ((null lst) Nil)
        ((= (length lst) 1) lst)
        (T
            (let ((f (car lst)))
                (setf (car lst) (car (last lst)))
                (setf (car (last lst)) f)
                lst
            )
        )
    )
)
```

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 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 roll ()
    (cons
        (+ (random 6) 1)
        (+ (random 6) 1)
    )
)
(defun check_reroll (pair)
    (or
        (= (car pair) (cdr pair) 1)
        (= (car pair) (cdr pair) 6)
    )
)
(defun roll_sum (pair)
    (+ (car pair) (cdr pair))
)
(defun check_win (pair)
    (or (= (roll_sum pair) 7)
         (= (roll_sum pair) 11)
    )
)
(defun play_round ()
    (let ( (pair (roll)) )
        (cond
            (
                (check_reroll pair)
                (play_round)
            )
            (T pair)
        )
    )
)
(defun play_dice ()
    (let ( (pl1 (play_round)) (pl2 (play_round)) )
        (format T "~%Player 1: ~A ~%" pl1)
        (format T "Player 2: ~A ~%" pl2)
        (cond
            (
                (check_win pl1)
                (princ "Player 1 wins")
            )
            (
                (check_win pl2)
                (princ "Player 2 wins")
            )
            (
                (> (roll_sum pl1) (roll_sum pl2))
                (princ "Player 1 wins")
            )
            (
                (< (roll_sum pl1) (roll_sum pl2))
                (princ "Player 2 wins")
            )
            (
                T
                (princ "Draw")
            )
        )
    )
)
```

Задание 7

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

```
(defun is_palindrome (lst)
    (cond
        ((< (length lst) 2) T)
        ((equalp (car lst) (car (last lst)))
        (is_palindrome (cdr (reverse (cdr lst)))))
        (T Nil)
    )
)
```

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.

```
(defun find_in_table (lst item)
    (cond
        ((null lst) Nil)
        ((equal item (caar lst)) (cdar lst))
        ((equal item (cdar lst)) (caar lst))
        (T (find_in_table (cdr lst) item))
    )
)
```

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,
б) элементы списка – любые объекты.

```
(defun mult_first_num (num lst)
 (cond
  ((null lst) Nil)
  ((numberp (car lst)) (cons (setf (car lst) (* (car lst) num)) (cdr lst)))
  (T (cons (car lst) (mult_first_num num (cdr lst))))
 )
)
```