**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 №1 по дисциплине «Функциональное и логическое программирование»

**Тема** Списки в Lispe. Использование стандартных функций.

**Студент** Завойских Е.В.

**Группа** ИУ7-63Б

**Оценка (баллы)** _____

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

Москва — 2023 г.

# 1 Теоретические вопросы

## 1.1 Элементы языка: определение, синтаксис, представление в памяти.

Вся информация (данные и программы) в Lisp представляется в виде символьных выражений — S-выражений. По определению

S-выражение ::= <атом>|<точечная пара>.

**Атомы**

- символы (идентификаторы) — синтаксически — набор литер (букв и цифр), начинающихся с буквы (например, ПримерАтома);
- 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”);

**Списки и точечные пары (структуры)**

Определения:

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>.

Синтаксически:

Любая структура (точечная пара или список) заключается в круглые скобки: (A . B) — точечная пара, (A) — список из одного элемента. Пустой список изображается как Nil или (); непустой список по определению может быть изображен: (A . (B . (C . (D . ())))), допустимо изображение списка последовательностью атомов, разделенных пробелами — (A B C D).

Элементы списка могут быть списками (любой список заключается в круглые скобки), например — (A (B C) (D C)). Таким образом, синтаксически наличие скобок является признаком структуры — списка или точечной пары.

Любая непустая структура Lisp в памяти представляется списковой ячейкой, хранящей два указателя: на голову (первый элемент) и хвост — всё остальное.

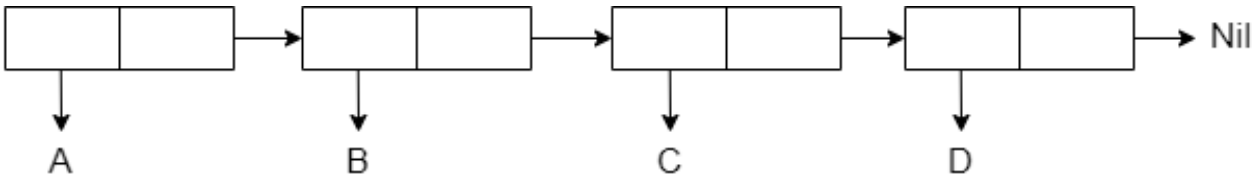


Рис. 1.1: Представление в памяти (A B C D)

## 1.2 Особенности языка Lisp. Структура программы. Символ апостроф.

Вся информация (данные и программы) в Lisp представляется в виде символьных выражений — S-выражений.

Lisp — язык символьной обработки. В Lisp программа и данные представлены списками. По умолчанию первый элемент списка трактуется как имя функции, а остальные — как ее аргументы.

Так как и программа, и данные представлены списками, то их нужно различать. Для этого была создана функция quote.

Функция quote блокирует вычисление своего аргумента. Апостроф — сокращённое обозначение функции quote. Перед самовычислимыми атомами (числами и атомами T, Nil) можно не ставить апостроф.

## 1.3 Базис языка Lisp. Ядро языка.

Базис — это минимальный набор конструкций языка, на основе которых могут быть построены остальные.

Базис Lisp:

- атомы;
- структуры;
- базовые функции и базовые функционалы (cons, car, cdr, eval, quote, atom, eq).

**Атомы**

- символы (идентификаторы) — синтаксически — набор литер (букв и цифр), начинающихся с буквы (например, ПримерАтома);

- 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”);

Более сложные данные — списки и точечные пары (структуры) строятся из унифицированных структур — блоков памяти — бинарных узлов.

# 2 Практические задания

## 2.1 Представить следующие списки в виде списочных ячеек

1. '(open close halph)

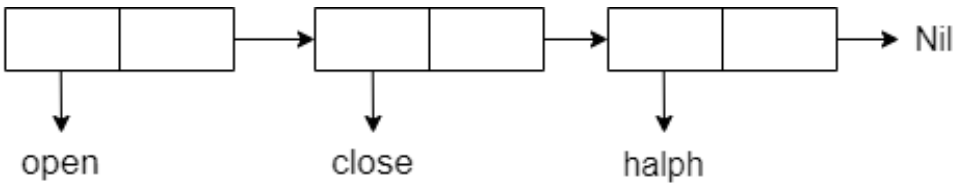


Рис. 2.1: Представление в памяти '(open close halph)

2. '((open1) (close2) (halph3))

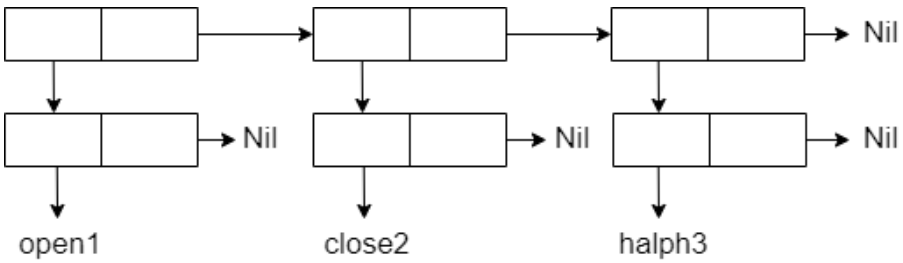


Рис. 2.2: Представление в памяти '((open1) (close2) (halph3))

3. '((one) for all (and (me (for you))))

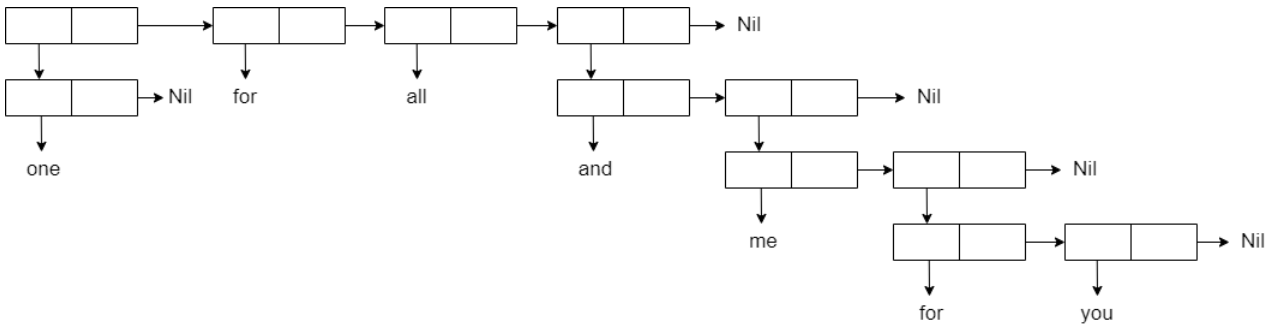


Рис. 2.3: Представление в памяти '((one) for all (and (me (for you))))

4. '((TOOL) (call))

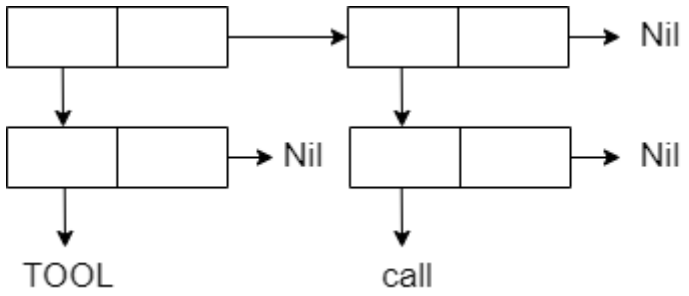


Рис. 2.4: Представление в памяти '((TOOL) (call))

5. '((TOOL1) ((call2)) ((sell)))

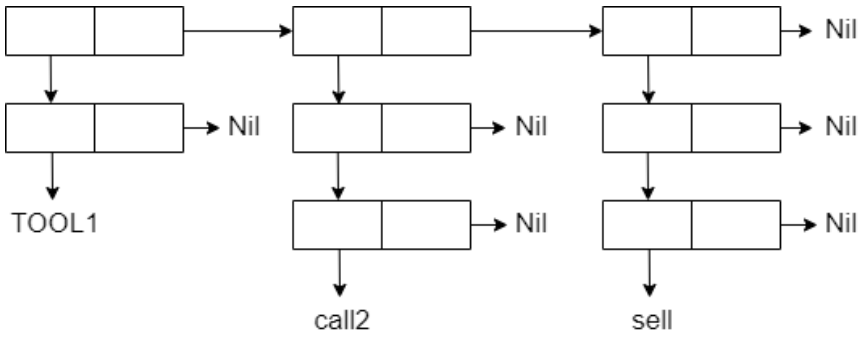


Рис. 2.5: Представление в памяти '((TOOL1) ((call2)) ((sell)))

6. '(((TOOL) (call)) ((sell)))

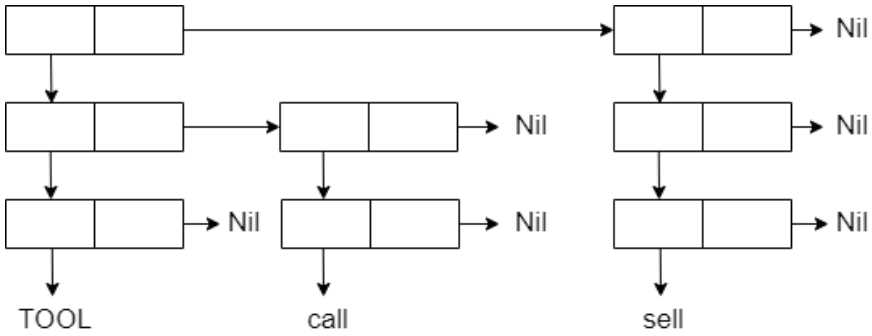


Рис. 2.6: Представление в памяти '(((TOOL) (call)) ((sell)))

## 2.2 Используя только функции CAR и CDR, написать выражения, возвращающие:

1. Второй элемент заданного списка

```
(CAR (CDR '(A B C D E))) -> B
```

2. Третий элемент заданного списка

```
(CAR (CDR (CDR '(A B C D E)))) -> C
```

3. Четвертый элемент заданного списка

```
(CAR (CDR (CDR (CDR '(A B C D E))))) -> D
```

## 2.3 Что будет в результате вычисления выражений?

1. (CAADR '((blue cube) (red pyramid))) -> red

```
(CDR '((blue cube) (red pyramid))) -> ((red pyramid))
(CAR '((red pyramid))) -> (red pyramid)
(CAR '(red pyramid)) -> red
```

2. (CDAR '((abc) (def) (ghi)) -> Nil

```
(CAR '((abc) (def) (ghi))) -> (abc)
(CDR '(abc)) -> Nil
```

3. (CADR '((abc) (def) (ghi))) -> (def)

```
(CDR '((abc) (def) (ghi))) -> ((def) (ghi))
(CAR '((def) (ghi))) -> (def)
```

4. (CADDR '((abc) (def) (ghi))) -> (ghi)

```
(CDR '((abc) (def) (ghi))) -> ((def) (ghi))
(CDR '((def) (ghi))) -> ((ghi))
(CAR '((ghi))) -> (ghi)
```

## 2.4 Напишите результат вычисления выражений и объясните, как он получен

quote блокирует вычисление своего аргумента. Перед числами и атомами T, Nil можно не ставить апостроф.

list принимает произвольное число аргументов. Функция создает и возвращает список, состоящий из значений аргументов.

cons имеет фиксированное количество аргументов (два). Если аргументами являются атомы, то создает точечную пару. Если первый аргумент — атом, а второй — список, атом становится головой, а второй аргумент (список) становится хвостом.

```
(list 'Fred 'and 'Wilma) -> (Fred and Wilma)
(list 'Fred '(and Wilma)) -> (Fred (and Wilma))
(cons Nil Nil) -> (Nil. Nil) -> (Nil)
(cons T Nil) -> (T. Nil) -> (T)
(cons Nil T) -> (Nil. T)
(list Nil) -> (Nil)
(cons '(T) Nil) -> ((T). Nil) -> ((T))
(list '(one two) '(free temp)) -> ((one two) (free temp))

(cons 'Fred '(and Willma)) -> (Fred and Willma)
(cons 'Fred '(Wilma)) -> (Fred Willma)
(list Nil Nil) -> (Nil Nil)
(list T Nil) -> (T Nil)
(list Nil T) -> (Nil T)
(cons T (list Nil)) -> (T Nil)
(list '(T) Nil) -> ((T) Nil)
(cons '(one two) '(free temp)) -> ((one two) free temp)
```

## 2.5 Написать лямбда-выражение и соответствующую функцию. Представить результаты в виде списочных ячеек.

1. Написать функцию (f ar1 ar2 ar3 ar4), возвращающую список: ((ar1 ar2) (ar3 ar4)).

```
(defun f (ar1 ar2 ar3 ar4) (list (list ar1 ar2) (list ar3 ar4)))
(lambda (ar1 ar2 ar3 ar4) (list (list ar1 ar2) (list ar3 ar4)))
```

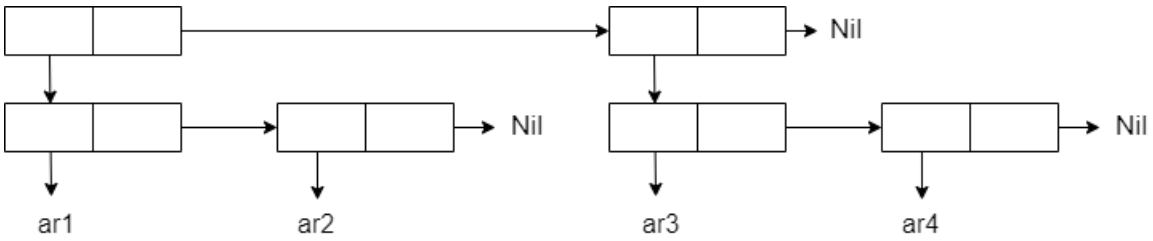


Рис. 2.7: Результаты первой функции

2. Написать функцию (f ar1 ar2), возвращающую ((ar1) (ar2)).

```
(defun f (ar1 ar2) (list (list ar1) (list ar2)))
(lambda (ar1 ar2) (list (list ar1) (list ar2)))
```

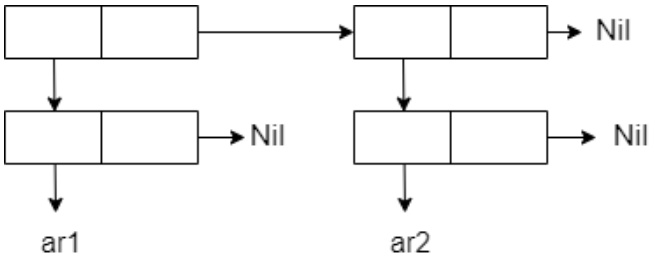


Рис. 2.8: Результаты второй функции

3. Написать функцию (f ar1), возвращающую (((ar1))).

```
(defun f (ar1) (list (list (list ar1))))
(lambda (ar1) (list (list (list ar1))))
```

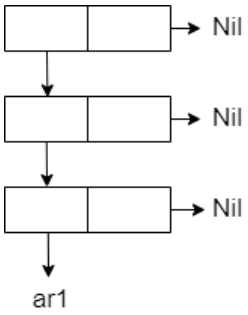


Рис. 2.9: Результаты третьей функции