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 №1 по курсу «Функциональное и логическое программирование»

Тема Списки в Lispe. Использование стандартных функций

Студент Динь Вьет Ань

Группа ИУ7И-64Б

Оценка (баллы) _____

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

Москва — 2023 г.

Теоретические вопросы

1. Элементы языка: определение, синтаксис, представление в памяти

Элементы языка

Элементами языка Lisp являются атомы и точечные пары (структуры). К атомам относятся:

- символы (идентификаторы) – набор литер, начинающихся с буквы;
- специальные символы для обозначения логических констант T, Nil;
- самоопределимые атомы – натуральные числа, дробные числа, вещественные числа, строки (последовательность символов, заключённых в двойные апострофы).

Синтаксис элементов языка и их представление в памяти.

Точечные пары ::= (<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>

Список – частный случай S-выражения.
Синтаксически любая структура (точечная пара или список) заключается в ():
(A . B) – точечная пара
(A) – список из одного элемента
Пустой список изображается как Nil или ()
Непустой список может быть изображён: (A. (B . (C ()))) или (A B C)
Элементы списка могут являться списками: ((A) (B) (C))
Любая непустая структура Lisp в памяти представлена списковой ячейкой, хранящей два указателя: на голову (первый элемент) и хвост (всё остальное).

2. Особенности языка LISP. Структура программы. Символ апостроф.

Структура программы

Lisp - язык символьной обработки. В Lisp программа и данные представлены списками. По умолчанию список считается вычислимой формой, в которой 1 элемент - название функции, остальные элементы - аргументы функции.

Особенности языка

Т.к. и программа и данные представлены списками, то их нужно как-то различать. Для этого была создана функция quote, а ' - ее сокращенное обозначение. quote - функция, блокирующая вычисление.

Символ апостроф

Символ `'` – функциональная блокировка, эквивалентен функции `quote`. Блокирует вычисление выражения. Таким образом, выражение воспринимается интерпретатором как данные.

3. Базис языка LISP. Ядро языка.

Базис – это минимальный набор инструментов языка и стркутур данных, который позволяет решить любые задачи.

Базис Lisp :

- атомы и структуры (представляющиеся бинарными узлами);
- базовые (несколько) функций и функционалов: встроенные — примитивные функции (atom, eq, cons, car, cdr); специальные функции и функционалы (quote, cond, lambda, eval, apply, funcall).

Функцией называется правило, по которому каждому значению одного или нескольких аргументов ставится в соответствие конкретное значение результата.

Функционалом, или функцией высшего порядка называется функция, аргументом или результатом которой является другая функция.

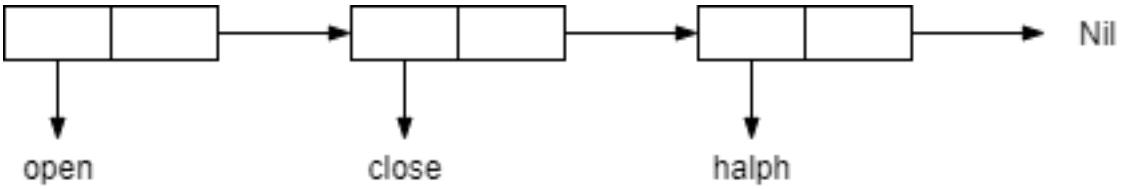
Некоторые функции системы необходимо реализовывать в виде машинных подпрограмм, они образуют ядро системы. Ядро - основные действия, которые наиболее часто используются. Понятие «ядро» шире, чем понятие «базис».

Практические задания

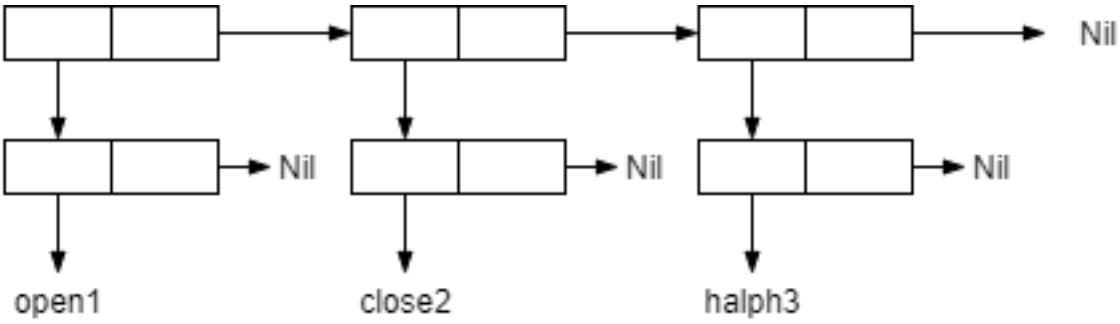
Задание 1

Представить следующие списки в виде списочных ячеек:

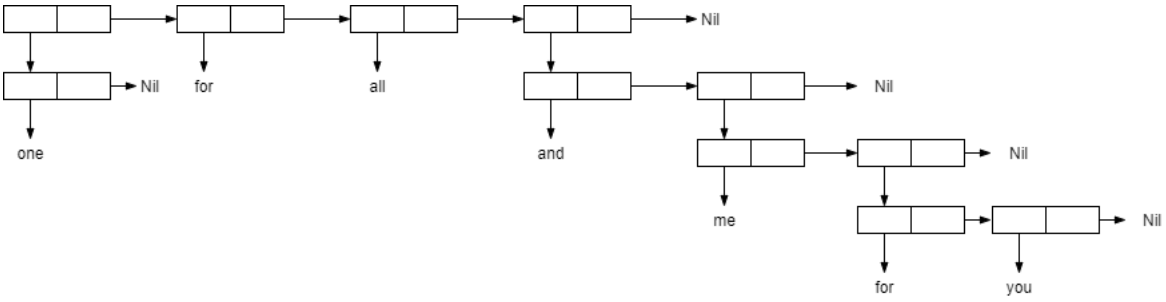
– `'(open close halp)`



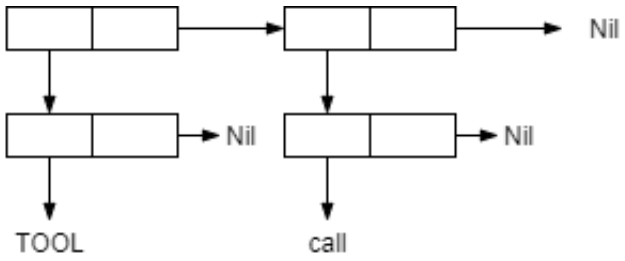
– `'((open1) (close2) (halp3))`



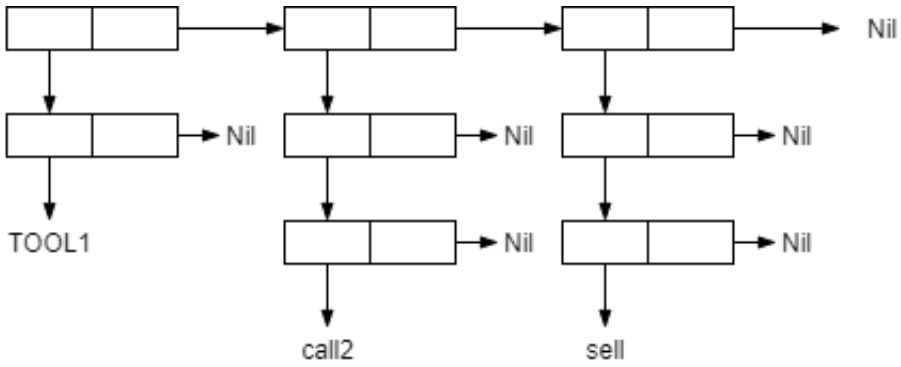
– `'((one) for all (and (me (for you))))`



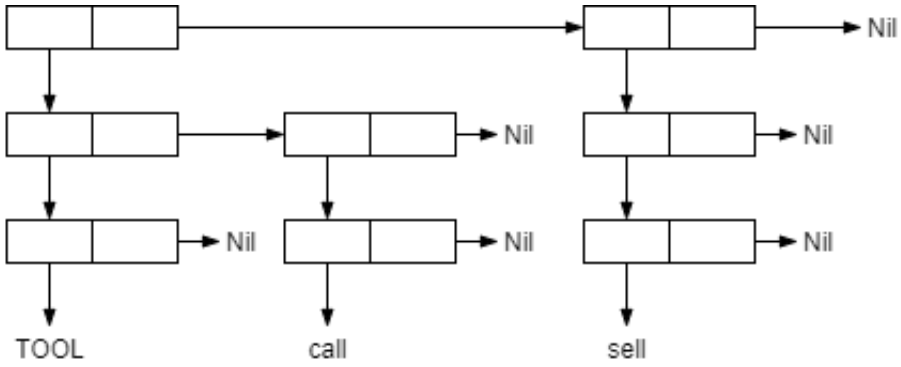
– `'((TOOL) (call))`



– `'((TOOL1) ((call2)) ((sell)))`



– `'(((TOOL) (call)) ((sell)))`



Задание 2

Используя только функции `CAR` и `CDR`, написать выражения, возвращающие второй, третий, четвёртый элементы заданного списка:

```
(CAR (CDR '(1 2 3 4 5))) => 2
(CADR '(1 2 3 4 5)) => 2

(CAR (CDR (CDR '(1 2 3 4 5)))) => 3
(CADDR '(1 2 3 4 5)) => 3

(CAR (CDR (CDR (CDR '(1 2 3 4 5))))) => 4
(CADDDR '(1 2 3 4 5)) => 4
```

Задание 3

Что будет в результате выполнения выражений?

```
(caadr '((blue cube)(red pyramid))) ; -> red
(cdar '((abc)(def)(ghi))) ; -> Nil
(cadr '((abc)(def)(ghi))) ; -> (def)
(caddr '((abc)(def)(ghi))) ; -> (ghi)
```

Задание 4

Напишите результат вычисления выражений:

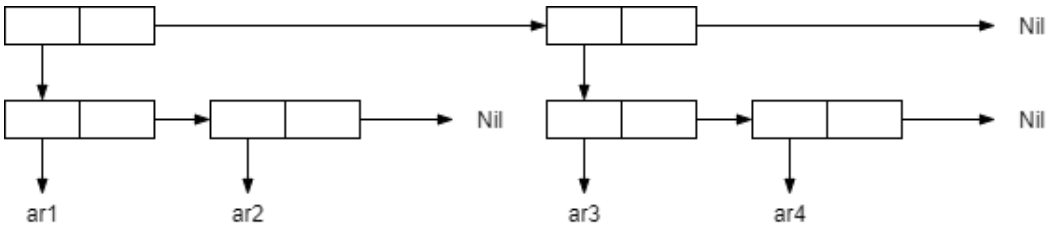
```
(list 'Fred 'and 'Wilma) => (Fred and Wilma)
(list 'Fred '(and Wilma)) => (Fred (and Wilma))
(cons Nil Nil) => (Nil . Nil) => (Nil)
(cons T Nil) => (T . Nil) => (T)
(cons Nil T) => (Nil . T)
(list Nil) => (Nil)
(cons '(T) Nil) => ((T) . Nil) => ((T))
(list '(one two) '(free temp)) => ((one two) (free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

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

Задание 5

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

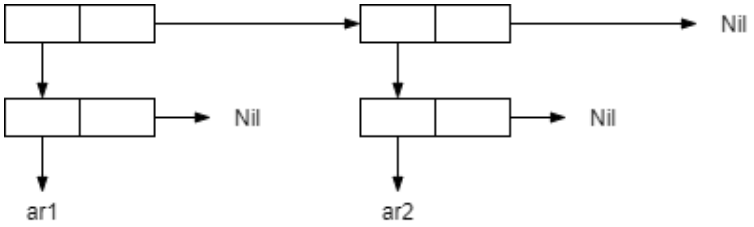
– Написать функцию (f arl ar2 ar3 ar4), возвращающую список ((arl ar2) (ar3 ar4)).

```
(defun f1 (ar1 ar2 ar3 ar4) (list (list ar1 ar2) (list ar3 ar4)))
(f1 1 2 3 4) => ((1 2) (3 4))
;;
(lambda (ar1 ar2 ar3 ar4) (list (list ar1 ar2) (list ar3 ar4)))
((lambda (ar1 ar2 ar3 ar4) (list (list ar1 ar2) (list ar3 ar4))) 1 2 3 4) => ((1
    2) (3 4))
```



– Написать функцию (f arl ar2), возвращающую ((arl) (ar2))

```
(defun f2 (ar1 ar2) (list (list ar1) (list ar2)))
(f2 1 2) => ((1) (2))
;;
(lambda (ar1 ar2) (list (list ar1) (list ar2)))
((lambda (ar1 ar2) (list (list ar1) (list ar2))) 1 2) => ((1) (2))\
```



– Написать функцию (f arl), возвращающую (((arl))).

```
(defun f3 (ar1) (list (list (list ar1))))
(f3 1) => (((1)))
;;
(lambda (ar1) (list (list (list ar1))))
((lambda (ar1) (list (list (list ar1)))) 1) => (((1)))
```