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»

Лабораторная работа № 1
по дисциплине «Функциональное и логическое программирование»

Тема Списки в Lispe. Использование стандартных функций.

Студент Калашников С.Д.

Группа ИУ7-63Б

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

Москва, 2023

СОДЕРЖАНИЕ

1 Ответы на вопросы

1.1 Элементы языка

Элементами языка являются атомы и точечные пары.
Атомы представляю из себя:

1. Символы — синтаксически представляется как набор букв и цифр, начинающийся с буквы.

2. Специальные символы — {T, Nil}.

3. Самоопределимые атомы — натуральные, дробные и вещественные числа, а также строки, заключенные в двойные апострофы.

   Атомы обычно выглядит как последовательность букв или цифр.
   Примеры атомов:

```
B
САТЯАтом
123ВотЭтоТожеАтом

123
T
Nil
2/3
"abc"
```

Точечная пара — (A . B). Строится с помощью бинарных узлов.

```
Точечнаяпара
        ::= атоматом(<>.<>) |атомточечная
    (<>.< пара>) |точечная
    (< параатом>.<>) |точечная
    (< параточечная>.< пара>)
```

Пример точечной пары:

```
(A . (B . (C . (D . Nil))))
```

Облегченная форма записи:

```
(A B C D)
```

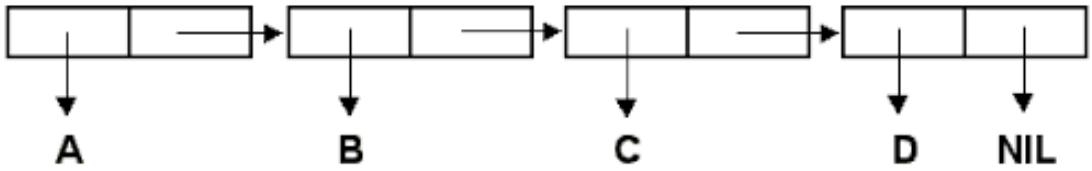

Рис. 1.1 – Представление в памяти (A B C D).

Представление в памяти:

1. **(A . B)**

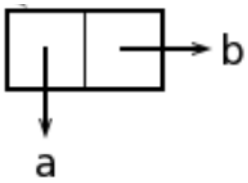

Рис. 1.2 – Представление в памяти (A . B).

2. **(A B)** — экономия памяти, но проблема при рекурсивной обработке (т.к. не сможем идентифицировать конец, как Nil)

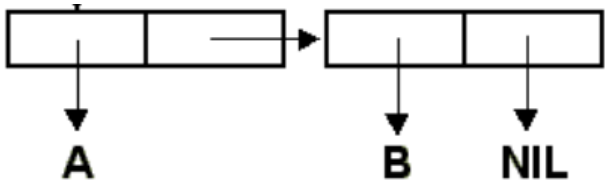

Рис. 1.3 – Представление в памяти (A B).

```
Sвыражение− ::= атом<> | точечная< пара>
```

Список является частым случаем S-выражения.

S-выражение представлено

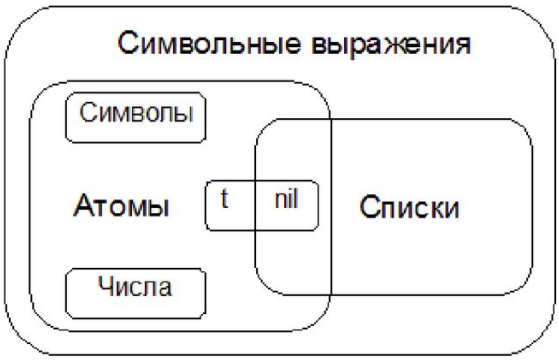

Рис. 1.4 – S-выражение

Список — динамическая структура данных, которая может быть пустой или непустой. Если она не пустая, то состоит из двух элементов:

1. Головы — любая структура.

2. Хвоста — список.

Список представляет из себя заключенную в скобки последовательность из атомов, разделенных пробелами, или списков. Любой список является программой - его нужно вычислять.

Примеры списков:

```
(A B C)
(1 2 3)
((A B) (C D))
```

Синтаксис

Lisp является регистронезависимым языком.

Универсальным разделителем, между атомами, является пробел. В начальных версиях была предложена запятая, но она не прижилась.

Наличие скобок является признаком структуры - списка или точечной пары.

Специальные символы:

1. **T** — Константа. обозначает логическое значение ”истина”. Истинным значением является все, что отличное от Nil.

2. **Nil** — ”ложь”. Также обозначает пустой список. Записи nil и () эквивалентны. Являются синтаксисом пустого списка

Любая структура заключается в круглые скобки.

(A . B) - точечная пара.

(A) - список из одного элемента.

() или Nil - пустой список.

Одноуровневый список:

```
(A B C D)
```

Структурированный список:

```
(A (B C) (D E))
```

Как воспринимается символ апостроф

Символ апостроф — синоним quote.

quote — блокирует вычисление своего аргумента. В качестве своего значения выдаёт сам аргумент, не вычисляя его. Перед константами - числами и атомами T, Nil можно не ставить апостроф.

Пример использования quote:

```
(quote (car (A B C))) => (car (A B C))
```

Вычисление начинается с внешней функции quote, которая возвращает аргумент в неизмененном виде.

1.2 Базис лиспа

Базис — минимальный набор средств для решения любой задачи.

Базис:

1) атомы и бинарные узлы;

2) atom, eq, cons, car, cdr, cond, quote, eval.

atom проверяет, является ли объект, переданный в качестве аргумента, атомом.

```
(atom 'a) ;; t
(atom '(a b c)) ;; nil
```

eq проверяет идентичность двух символов.

```
(eq 'a 'b) ;; nil
(eq 'a 'a) ;; t
```

cond — сокращение от англ. condition — условие. Не имеет фиксированного количества аргументов. Каждый аргумент — это список, голова которого рассматривается как условие, и если оно истинно, то результатом будет хвост рассматриваемого списка.

```
(cond ((eq 'A 'B) 'are_equal)
(T 'not_equal)) ;; NOT_EQUAL
```

eval - выполняет двойное вычисление своего аргумента.

```
(eval (cons (quote car) (quote ('(A B))))) => A
|---------(car '(A B))---------|
```

Функции car и cdr

car и cdr - базовые функции доступа к данным.

car — принимает точечную пару или список и возвращает голову (первый элемент).

cdr — принимает точечную пару или список и возвращает хвост (все элементы, кроме первого).

Отличие list и cons

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

```
(cons 'a 'b)      ;; (A . B)
(cons 'a '(a b c))   ;; (A A B C)
(cons '(a c) '(b d)) ;;((A C) B D)
(cons 'a 'v 'd)    ;; Error (invalid number of arguments: 3)
```

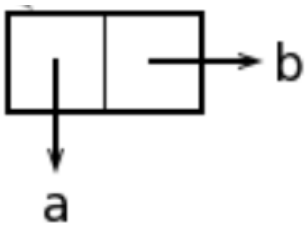

Рис. 1.5 – Результат cons.

list — не имеет фиксированное количество аргументов. Создает список, у которого голова — это первый аргумент, хвост — все остальные аргументы.

```
(list 'a 'b)              ;; (A B)
(list 'a 'b 'v '(c d) 'd)    ;; (A B V (C D) D)
```

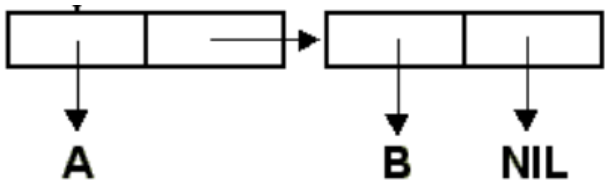

Рис. 1.6 – Результат list.

cons — имеет фиксированное число аргументов и более экономный по памяти.

Ядро — основные действия, которые наиболее часто используются. Ядро шире, чем базис.

2 Практическая часть

2.1 Задание 1

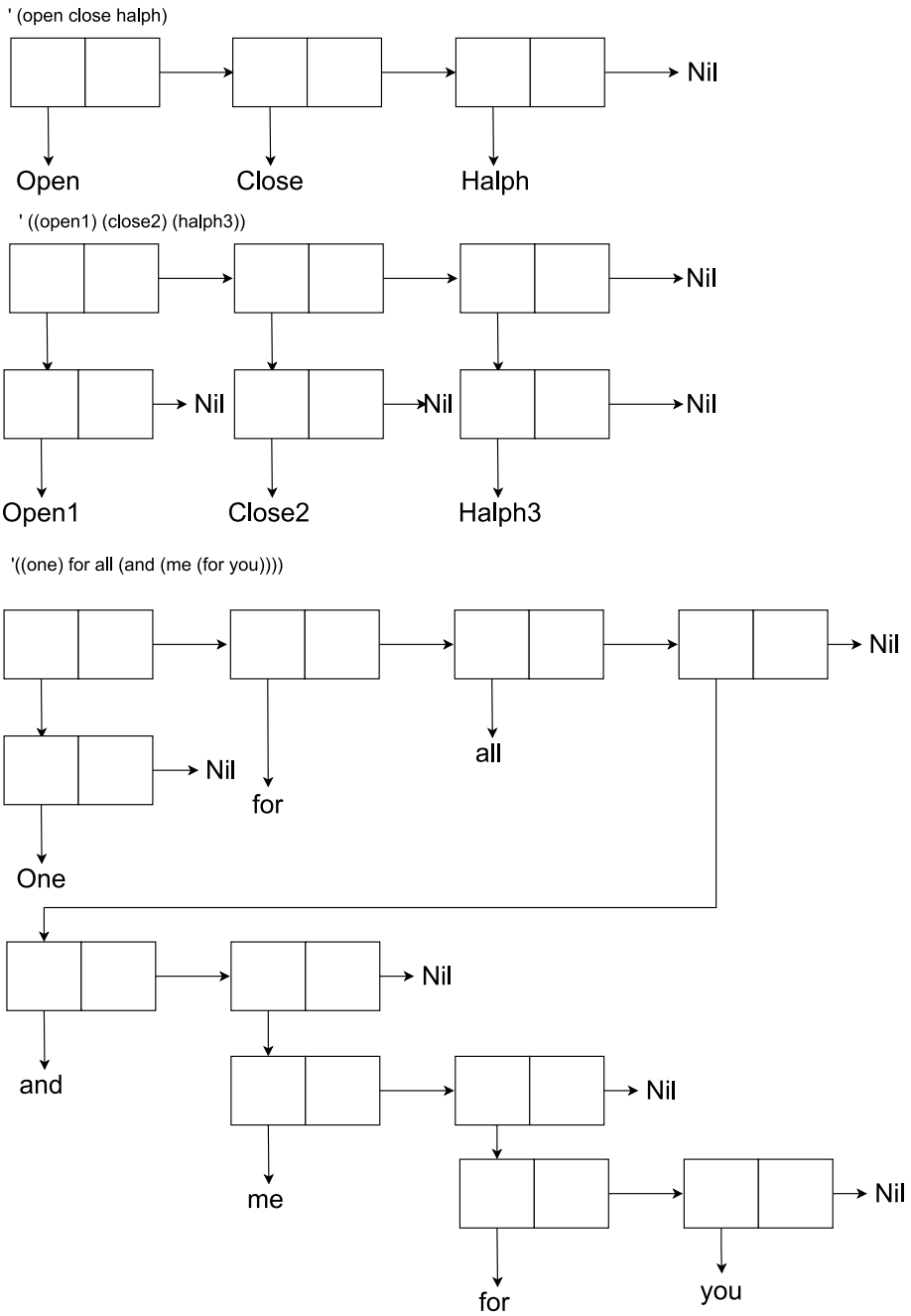

Рис. 2.1 – Часть 1

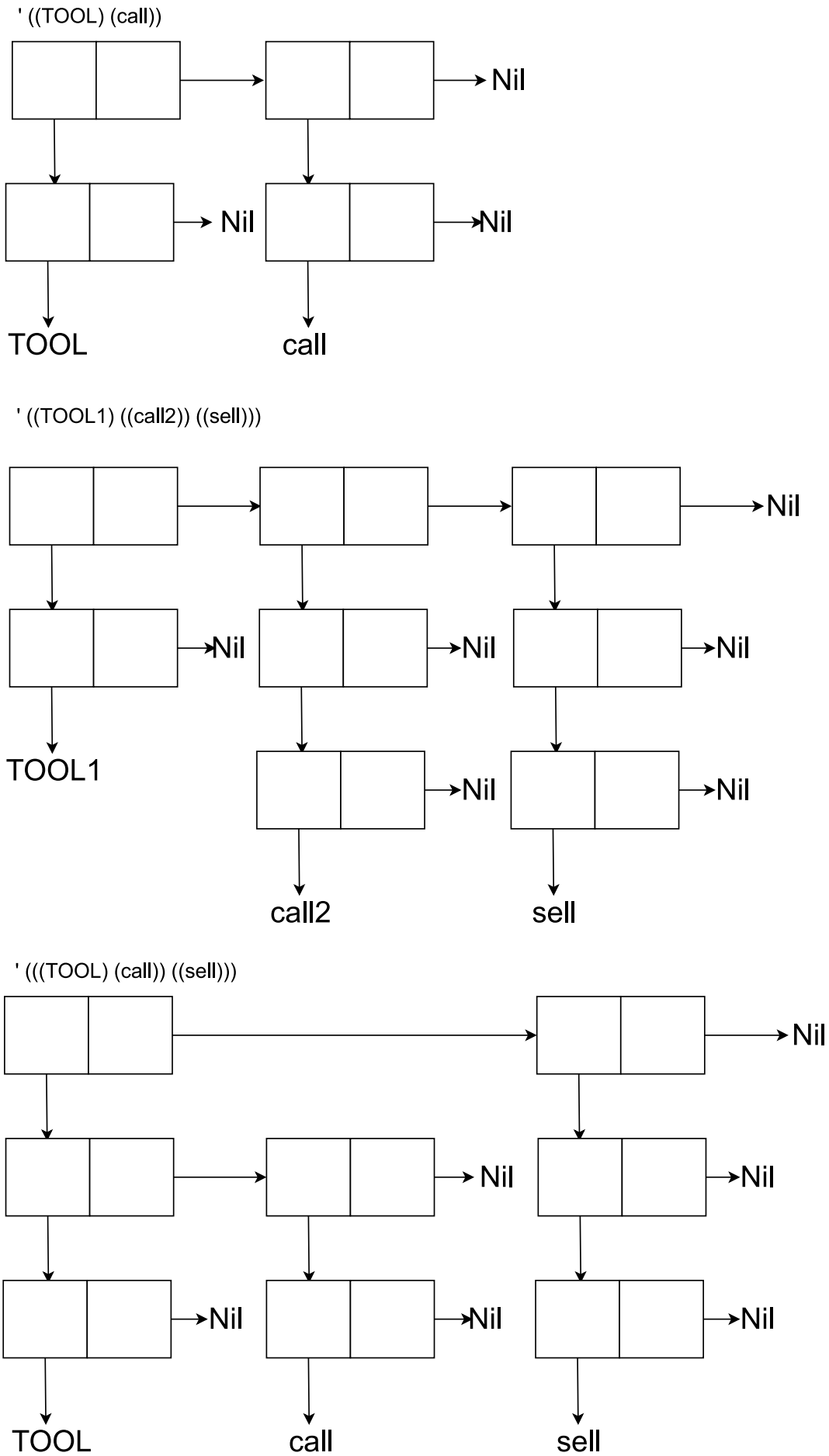

Рис. 2.2 – Часть 2

2.2 Задание 2

Листинг 2.1 — Выражение 1

```
(CAR (CDR '(1 2 3 4 5 6)))
```

Листинг 2.2 — Выражение 2

```
(CAR (CDR (CDR '(1 2 3 4 5 6))))
```

Листинг 2.3 — Выражение 3

```
(CAR (CDR (CDR (CDR '(1 2 3 4 5 6)))))
```

2.3 Задание 3

Листинг 2.4 — Выражение 1

```
(CAADR ' ((blue cube) (red pyramid)))  ;=red
```

Листинг 2.5 — Выражение 2

```
(CDAR '((abc) (def) (ghi)))  ;=nil
```

Листинг 2.6 — Выражение 3

```
(CADR ' ((abc) (def) (ghi)))  ;=def
```

Листинг 2.7 — Выражение 4

```
(CADDR ' ((abc) (def) (ghi)))  ;=ghi
```

2.4 Задание 4

Листинг 2.8 — Выражение 1

```
(list 'Fred 'and 'Wilma)  ;(FRED AND WILMA)
```

Листинг 2.9 — Выражение 2

```
(list 'Fred '(and Wilma))  ;(FRED (AND WILMA))
```

Листинг 2.10 — Выражение 3

```
(cons Nil Nil) ;(NIL)
```

Листинг 2.11 — Выражение 4

```
(cons T Nil) ;(T)
```

Листинг 2.12 — Выражение 5

```
(cons Nil T) ;(NIL . T)
```

Листинг 2.13 — Выражение 6

```
(list Nil) ;(NIL)
```

Листинг 2.14 — Выражение 7

```
(cons ' (T) Nil) ;((T))
```

Листинг 2.15 — Выражение 8

```
(list ' (one two) ' (free temp)) ;((ONE TWO) (FREE TEMP))
```

Листинг 2.16 — Выражение 9

```
(cons 'Fred '(and Wilma)) ;(FRED AND WILMA)
```

Листинг 2.17 — Выражение 10

```
(cons 'Fred '(Wilma)) ;(FRED WILMA)
```

Листинг 2.18 — Выражение 11

```
(list Nil Nil) ;(NIL NIL)
```

Листинг 2.19 — Выражение 12

```
(list T Nil) ;(T NIL)
```

Листинг 2.20 — Выражение 13

```
(list Nil T) ;(NIL T)
```

Листинг 2.21 — Выражение 14

```
(cons T (list Nil)) ;(T NIL)
```

Листинг 2.22 — Выражение 15

```
(cons '(one two) '(free temp)) ;((ONE TWO) FREE TEMP)
```

Листинг 2.23 — Выражение 16

```
(list '(T) Nil) ;((T) NIL)
```

2.5 Задание 5

Листинг 2.24 — Выражение 1

```
(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)))
    1 2 3 4)
```

Листинг 2.25 — Выражение 2

```
(defun f (ar1 ar2) (list (list ar1) (list ar2)))
((lambda (ar1 ar2) (list (list ar1) (list ar2))) 1 2)
```

Листинг 2.26 — Выражение 3

```
(defun f (ar1) (list (list (list ar1))))
((lambda (ar1) (list (list (list ar1)))) 1)
```