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 дисциплине «Функциональное и логическое программирование»

Тема: Списки в Lisp. Использование стандартных функций.
Студент: Карпова Е. О.
Группа: ИУ7-62Б
Оценка (баллы): _____
Преподаватели: Толпинская Н. Б., Строганов Ю. В.

Москва — 2023 г.

Оглавление

1. Теоретическая часть **3**

1.1. Элементы языка: определение, синтаксис, представление в памяти. 3

1.2. Особенности языка Lisp. Структура программы. Символ апостроф. 5

1.3. Базис языка Lisp. Ядро языка. 5

2. Практическая часть **6**

2.1. Задание №1 6

2.2. Задание №2 6

2.3. Задание №3 6

2.4. Задание №4 7

2.5. Задание №5 8

1. Теоретическая часть

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

Элементами языка Lisp являются атомы и структуры (такие как список и точечная пара). Вся информация (данные и программы) в Lisp представляется в виде символьных выражений — S-выражений.

$$S - \text{выражение} ::= < \text{атом} > | < \text{точечная пара} > . \tag{1.1}$$

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 (например 2/3), вещественные числа, строки — последовательность символов, заключенных в двойные апострофы (например “abc”).

Самоопределимые атомы нельзя переопределить или создать функцию с именем самоопределимого атома. Специальные символы нельзя переопределить, но можно создать функцию с именем NIL. Символам можно присваивать значение переменной и функции.

Структуры строятся из бинарных узлов. Структуры:

$$\begin{aligned} &\text{точечная пара} ::= (< \text{атом} > . < \text{атом} >)|(< \text{атом} > . < \text{точечная пара} >) \\ &|(< \text{точечная пара} > . < \text{атом} >)|(< \text{точечная пара} > . < \text{точечная пара} >), \end{aligned} \tag{1.2}$$

$$\text{список} ::= < \text{пустой список} > | < \text{непустой список} >, \tag{1.3}$$

$$< \text{пустой список} >::= ()|Nil, \tag{1.4}$$

$$< \text{непустой список} >::= (< \text{первый элемент} > . < \text{хвост} >), \tag{1.5}$$

$$< \text{первый элемент} >::=< \text{S-выражение} >, \tag{1.6}$$

$$< \text{хвост} >::=< \text{список} > . \tag{1.7}$$

Точечная пара является более общей структурой чем список (список является частным случаем точечной пары). Любая непустая структура Lisp в памяти представляется списковой ячейкой, хранящей два указателя: на голову (первый элемент) и хвост — все остальное.

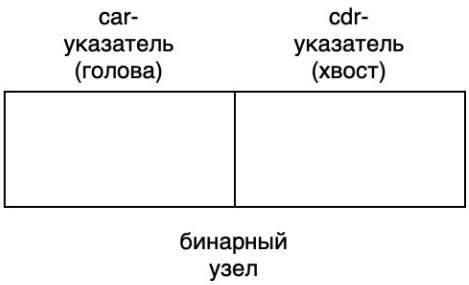

Рисунок 1.1 — Бинарный узел

Синтаксис:

— любая структура 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 как (A . (B . (C . (D ())))), допустимо изображение списка последовательностью атомов, разделенных пробелами — (A B C D);

— элементы списка могут, в свою очередь, быть списками, например — (A (B C) (D (E))).

Таким образом, синтаксически наличие скобок является признаком структуры — списка или точечной пары.

Точечная пара — это пара атомов или точечных пар или их комбинация.

Список — это структура данных. Может быть пустой и непустой. Если непустой, то состоит из двух элементов: первый (голова) — любая структура (нет слова тип), а второй (хвост) — список. По умолчанию первый аргумент списка — имя функции, остальные — аргументы функции.

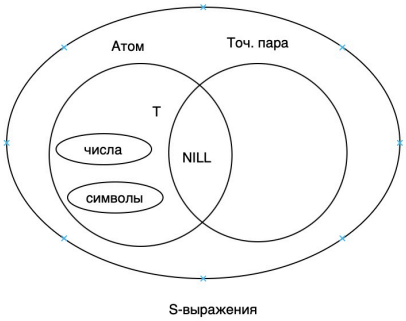

Рисунок 1.2 — Множества элементов языка

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

Программы, написанные на Lisp, представляются в виде его же структур данных, то есть синтаксического отличия нет. Таким образом, можно выдать программы за данные или наоборот, то есть программа способна менять сама себя. Программа на языке Lisp представляет собой последовательность вычислимых выражений, которые являются атомами или списками. Символ ' (апостроф) эквивалентен функции *quote* — блокирует вычисление выражения, и оно воспринимается интерпретатором как данные. Также, Lisp — бестиповой язык, поэтому выделяемые под элементы объёмы памяти унифицированы.

1.3. Базис языка Lisp. Ядро языка.

Базис — это минимальный набор правил/конструкций языка, к которым могут быть сведены все остальные. Базис языка Lisp представлен атомами, структурами, базовыми функциями, базовыми функционалами. Некоторые базисные функции: *car*, *cdr*, *cons*, *list*, *lambda*, *quote*, *eq*, *eql*, *equal*, *eval*, *apply*, *funcall*. Ядром языка называется базис языка и функции стандартной библиотеки (часто используемые функции, созданные на основе базиса).

Этапы обработки выражения:

1. ожидание ввода s-выражения;
2. передача его функции *eval*, она вычисляет значение переданного аргумента и возвращает его, как результат;
3. вывод полученного результата.

2. Практическая часть

2.1. Задание №1

Решено на первом занятии. Решение приложено к отчёту.

2.2. Задание №2

Используя только функции CAR и CDR, написать выражения, возвращающие второй, третий, четвертый элементы заданного списка.

CAR принимает точечную пару или список в качестве аргумента и возвращает первый элемент (голову) или NIL, если передан пустой список. CDR принимает точечную пару или список и возвращает хвост. В случае аргумента-непустого списка будет возвращен список, состоящий из всех элементов, кроме первого. В случае аргумента-пустого списка возвращается NIL.

Листинг 2.1 — Задание №2

```
; Получить i-ый элемент списка
(CAR (CDR '(1 2 3 4))) ; 2-ой
(CAR (CDR (CDR '(1 2 3 4)))) ; 3-ий
(CAR (CDR (CDR (CDR '(1 2 3 4))))) ; 4-ый
```

2.3. Задание №3

Что будет в результате вычисления выражений? (ответ написан в комментарии после каждого выражения).

Пояснение:

— (CAADR ' ((blue cube) (red pyramid))) == (CAR (CAR (CDR '((blue cube) (red pyramid)))))

— (CDAR '((abc) (def) (ghi))) == (CDR (CAR '((abc) (def) (ghi))))

— (CADR '((abc) (def) (ghi))) == (CAR (CDR '((abc) (def) (ghi))))

— (CADDR '((abc) (def) (ghi))) == (CAR (CDR (CDR '((abc) (def) (ghi)))))

Листинг 2.2 — Задание №3

```
(CAADR '((blue cube) (red pyramid))) ; RED
(CDAR '((abc) (def) (ghi))) ; NIL
(CADR '((abc) (def) (ghi))) ; (DEF)
(CADDR '((abc) (def) (ghi))) ; (GHI)
```

2.4. Задание №4

Напишите результат вычисления выражений (ответ написан в комментарии после каждого выражения).

Пояснение:

— Функция *list* возвращает список, содержащий значения переданных ей аргументов. Элемент с апострофом интерпретатор воспринимает как данные.

— Функция *cons* создает списковую ячейку с первым аргументом-головой и вторым-хвостом. Первый аргумент *cons* может быть любым из допустимых в языке, а второй обязательно список. Если второй аргумент является атомом, то создаётся точечная пара. Если второй аргумент является списком, то возвращается список, объединяющий предыдущие элементы с элементами второго аргумента-списка.

Листинг 2.3 — Задание №4

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

(cons T Nil) ; (T)

(cons Nil T) ; (NIL . T)

(list Nil) ; (NIL)

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

Листинг 2.4 — Задание №4

```
(list ' (one two) ' (free temp)) ; ((ONE TWO) (FREE TEMP))
(cons 'Fred '(and Wilma)) (cons 'Fred '(Wilma)) (list Nil Nil)
; (FRED AND WILMA) (FRED WILMA) (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. Задание №5

Написать лямбда-выражение и соответствующую функцию: Написать функцию (f arl ar2 ar3 ar4), возвращающую список: ((arl ar2) (ar3 ar4)). Написать функцию (f arl ar2), возвращающую ((arl) (ar2)). Написать функцию (f arl), возвращающую (((arl))). Представить результаты в виде списочных ячеек.

Функция — однозначное отображение множества значений аргументов в значение функции. Базисные функции — предоставляются языком, минимально необходимые для работы языка.

По аргументам и поведению:

— Базовые/чистые функции — фиксированное количество аргументов, для определенного набора аргументов один фиксированный результат;

— Формы — функции, которые принимают произвольное количество аргументов или по разному обрабатывают аргументы;

— Функционалы (высшего порядка) — в качестве аргумента принимают функцию или возвращают функцию.

По именованию:

— Именованные — имеют имя, можно определить через defun. Специальные символы (T, NIL) и самоопределимые атомы (натуральные, вещественные числа, строки) не могут выступать в роли функции;

— Неименованные — нет имени, определяются с помощью *lambda*.

Листинг 2.5 — Задание №5

```
; Задание: (f ar1 ar2 ar3 ar4) => ((ar1 ar2) (ar3 ar4))
; лямбда-выражение
((lambda (ar1 ar2 ar3 ar4) (list (list ar1 ar2) (list ar3 ar4))) 1 2 3 4)
; функция
(defun f1 (ar1 ar2 ar3 ar4) (list (list ar1 ar2) (list ar3 ar4)))
; вызов функции
(f1 1 2 3 4)
```

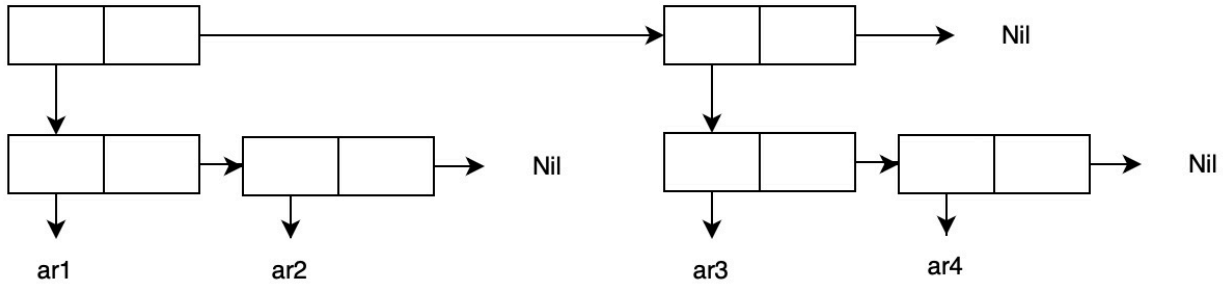

Рисунок 2.1 — Результат 1

Листинг 2.6 — Задание №5

```
; Задание: (f ar1 ar2) => ((ar1) (ar2))
; лямбда-выражение
((lambda (ar1 ar2) (list (list ar1) (list ar2))) 1 2)
; функция
(defun f2 (ar1 ar2) (list (list ar1) (list ar2)))
; вызов функции
(f2 1 2)
```

Листинг 2.7 — Задание №5

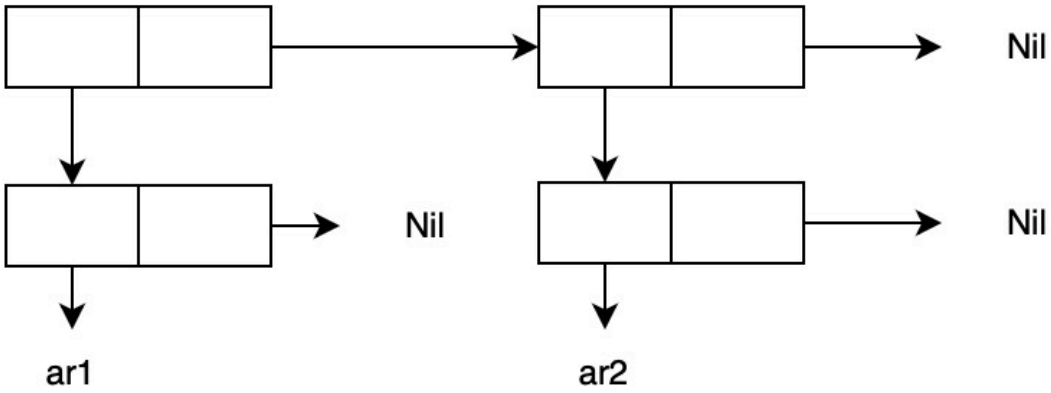

Рисунок 2.2 — Результат 2

```
; Задание: (f arl) => (((arl)))
; лямбда-выражение
((lambda (ar1) (list (list (list ar1)))) 1)
; функция
(defun f3 (ar1) (list (list (list ar1))))
; вызов функции
(f3 1)
```

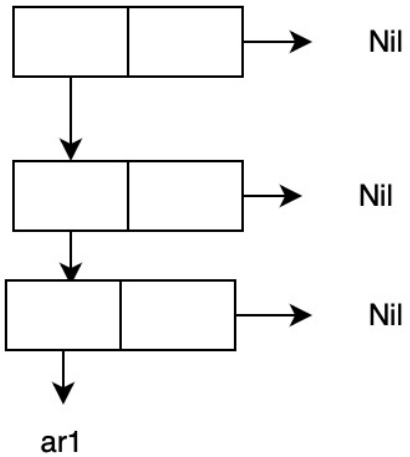

Рисунок 2.3 — Результат 3