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 курсу «Функциональное и логическое программирование»

на тему: «Списки в Lisp'е. Использование стандартных функций.»

Студент ИУ7-65Б _____ Турчанинов А. М.
(Группа) (Подпись, дата) (И. О. Фамилия)

Преподаватель _____ Толпинская Н. Б.
(Подпись, дата) (И. О. Фамилия)

2024 г.

1 Практическая часть

1.1 Используя только функции CAR и CDR, написать выражения, возвращающие: (№2)

1.1.1 Второй элемент заданного списка

Листинг 1.1 – Выражение, возвращающее второй элемент заданного списка

```
1 (car (cdr '(A B C D)))
```

1.1.2 Третий элемент заданного списка

Листинг 1.2 – Выражение, возвращающее третий элемент заданного списка

```
1 (car (cdr (cdr '(A B C D))))
```

1.1.3 Четвертый элемент заданного списка

Листинг 1.3 – Выражение, возвращающее четвертый элемент заданного списка

```
1 (car (cdr (cdr (cdr '(A B C D)))))
```

1.2 Что будет в результате вычисления выражений? (№3)

Листинг 1.4 – Выражение 1

```
1 (caadr '((blue cube) (red pyramid)))
```

Результат: `red`.

Листинг 1.5 – Выражение 2

```
1 (cdar '((abc) (def) (ghi)))
```

Результат: `Nil`

Листинг 1.6 – Выражение 3

```
1 (cadr '((abc) (def) (ghi)))
```

Результат: `(def)`

Листинг 1.7 – Выражение 4

```
1 (caddr '((abc) (def) (ghi)))
```

Результат: `(ghi)`

1.3 Что будет в результате вычисления выражении? (№4)

Таблица 1.1 – Результаты вычисления выражений

Выражение	**Результат**
`(list 'Fred 'and 'Wilma)`	`(Fred and Wilma)`
`(cons 'Fred '(and Wilma))`	`(Fred and Wilma)`
`(list 'Fred '(and Wilma))`	`(Fred (and Wilma))`
`(cons 'Fred '(Wilma))`	`(Fred Wilma)`
`(cons Nil Nil)`	`(Nil)`
`(list Nil Nil)`	`(Nil Nil)`
`(cons T Nil)`	`(T)`
`(list T Nil)`	`(T Nil)`
`(cons Nil T)`	`(Nil . T)`
`(list Nil T)`	`(Nil T)`
`(list Nil)`	`(Nil)`
`(cons T (list Nil))`	`(T Nil)`
`(cons '(T) Nil)`	`((T))`
`(list '(T) Nil)`	`((T) Nil)`
`(list '(one two) '(free temp))`	`((one two) (free temp))`
`(cons '(one two) '(free temp))`	`((one two) free temp)`

1.4 Написать lambda-выражение и соответствующую функцию возвращающие список... (№5)

- Функция `(f ar1 ar2 ar3 ar4)`, возвращающая список следующего вида:
  `((ar1 ar2) (ar3 ar4))`;

Листинг 1.8 – lambda-выражение

```
1    (lambda (ar1 ar2 ar3 ar4)
2        (list (list ar1 ar2) (list ar3 ar4))
3    )
```

Листинг 1.9 – функция, использующая `list`

```
1    (defun f (ar1 ar2 ar3 ar4)
2        (list (list ar1 ar2) (list ar3 ar4))
3    )
```

Листинг 1.10 – функция, использующая `cons`

```
1    (defun f (ar1 ar2 ar3 ar4)
2        (cons
3            (cons ar1 (cons ar2 Nil))
4            (cons
5                (cons ar3 (cons ar4 Nil))
6                Nil
7            )
8        )
9    )
```

- Функция `(f ar1 ar2)`, возвращающая список следующего вида:
  `((ar1) (ar2))`;

Листинг 1.11 – lambda-выражение

```
1    (lambda (ar1 ar2) (list (list ar1) (list ar2)))
```

Листинг 1.12 – функция, использующая `list`

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

Листинг 1.13 – функция, использующая `cons`

```
1     (defun f (ar1 ar2)
2         (cons
3             (cons ar1 Nil)
4             (cons (cons ar2 Nil) Nil)
5         )
6     )
```

- Функция `(f ar1)`, возвращающая список следующего вида: `(((ar1)))`;

Листинг 1.14 – lambda-выражение

```
1     (lambda (ar1) (list (list (list ar1))))
```

Листинг 1.15 – функция, использующая `list`

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

Листинг 1.16 – функция, использующая `cons`

```
1     (defun f (ar1) (cons (cons (cons ar1 Nil) Nil) Nil))
```

- Представить результаты в виде списочных ячеек:

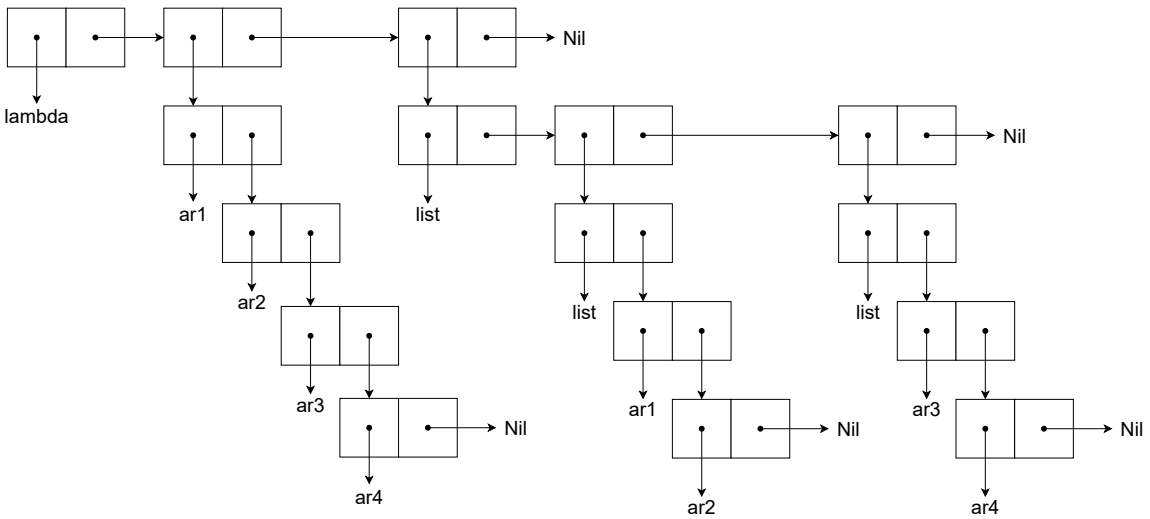

Рисунок 1.1 – Списочные ячейки для листинга 1.8

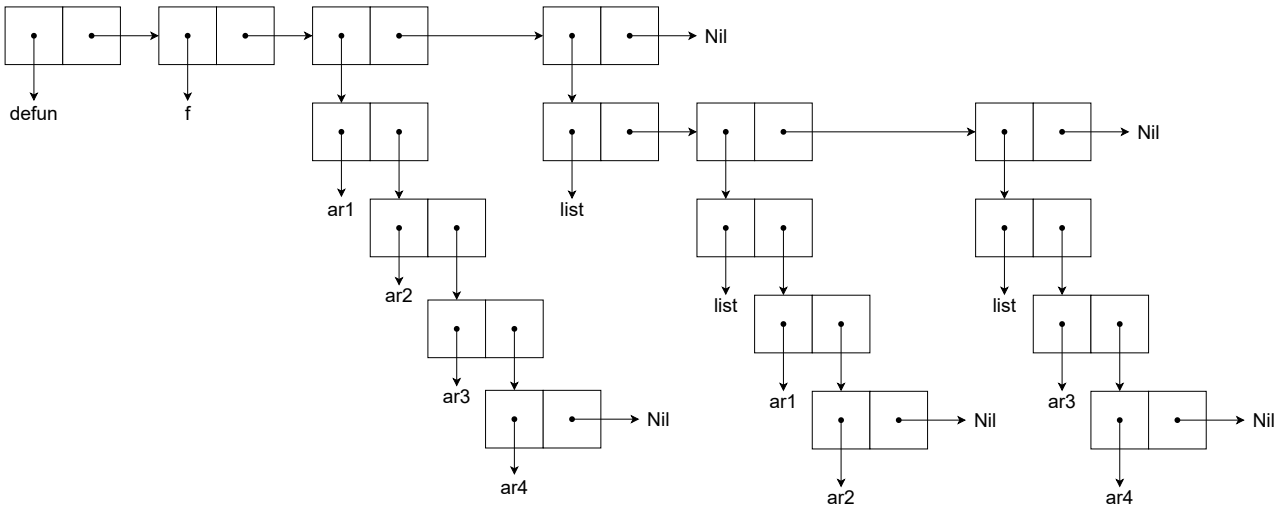

Рисунок 1.2 – Списочные ячейки для листинга 1.9

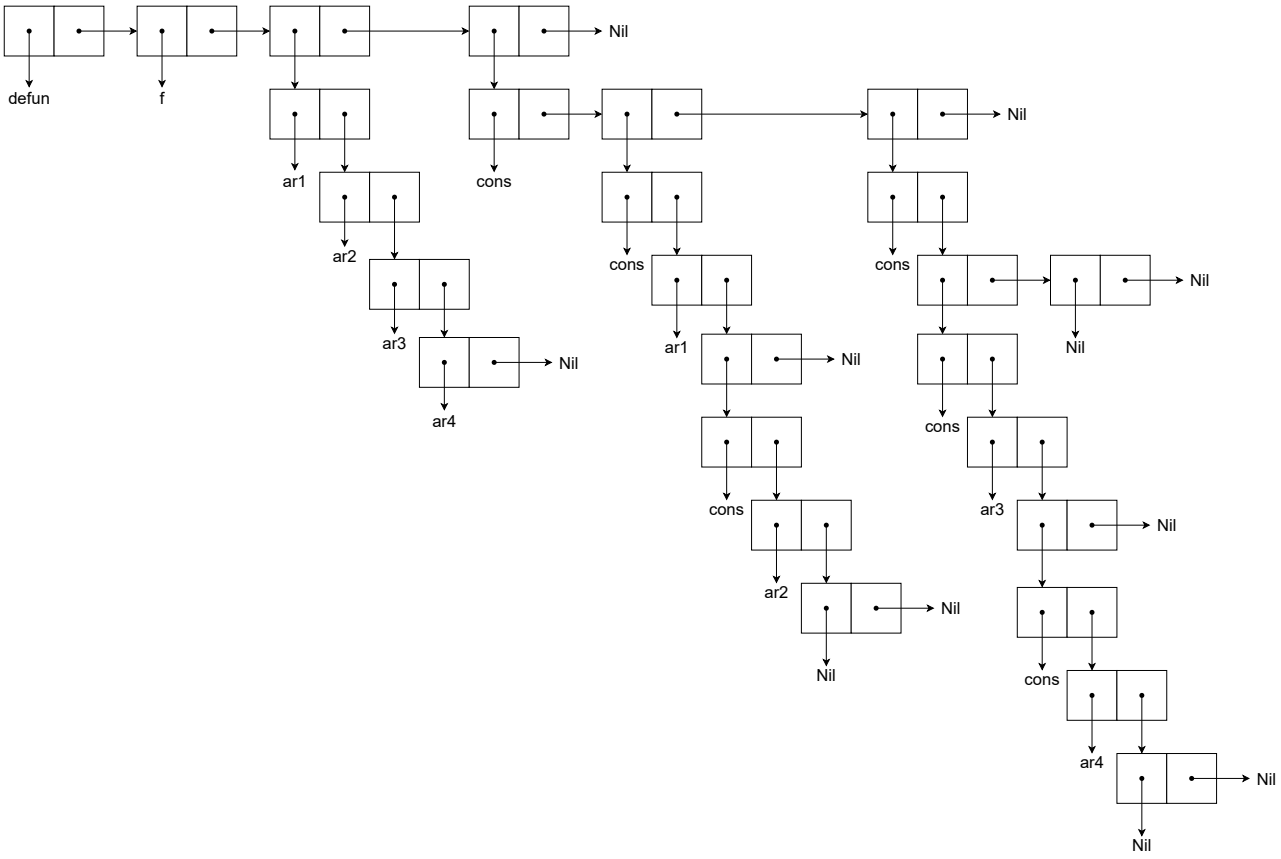

Рисунок 1.3 – Списочные ячейки для листинга 1.10

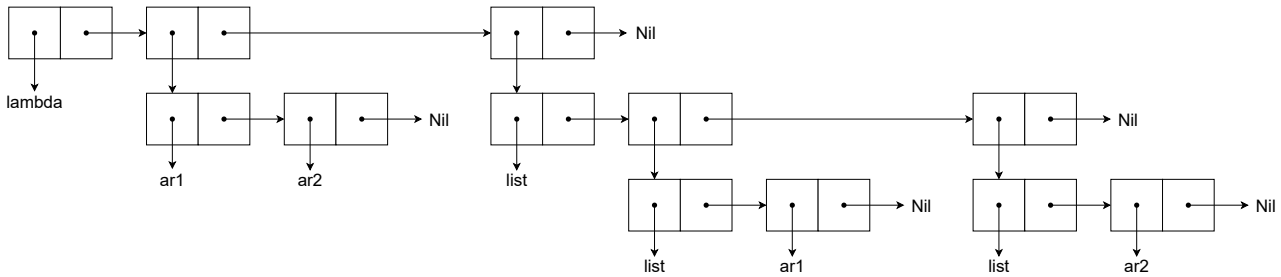

Рисунок 1.4 – Списочные ячейки для листинга 1.11

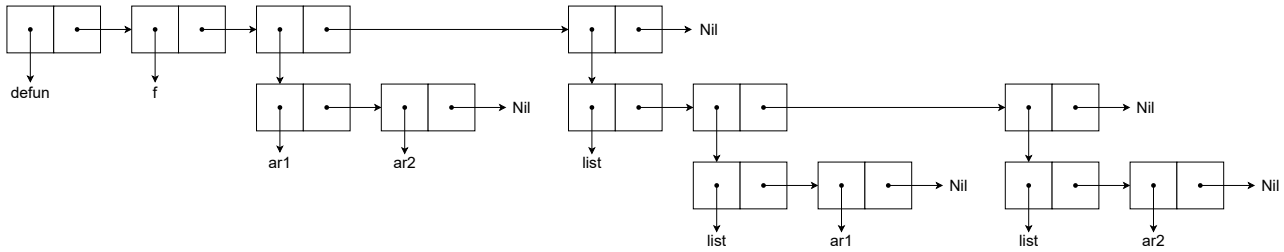

Рисунок 1.5 – Списочные ячейки для листинга 1.12

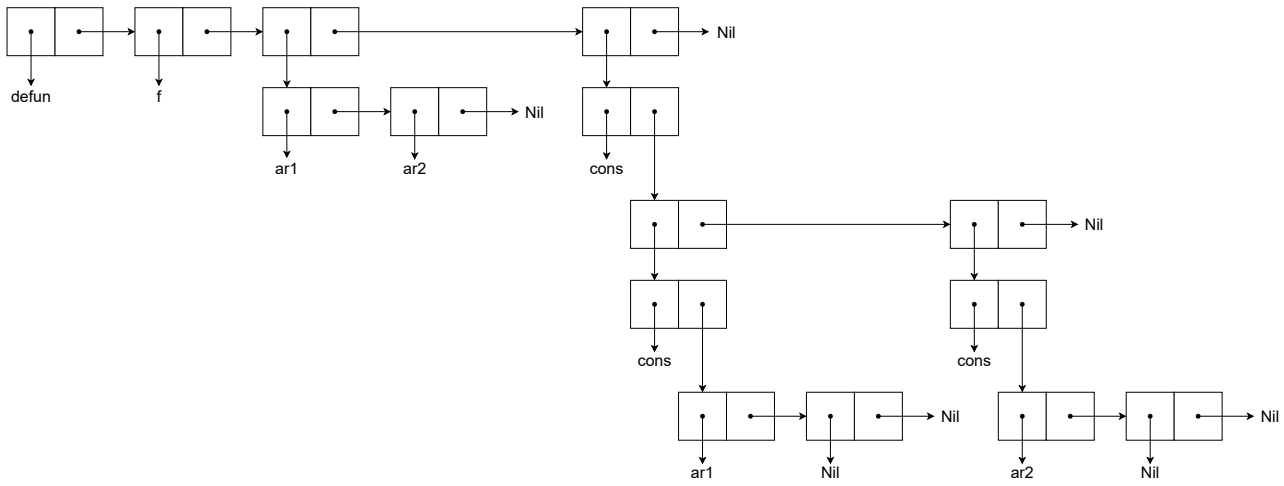

Рисунок 1.6 – Списочные ячейки для листинга 1.13

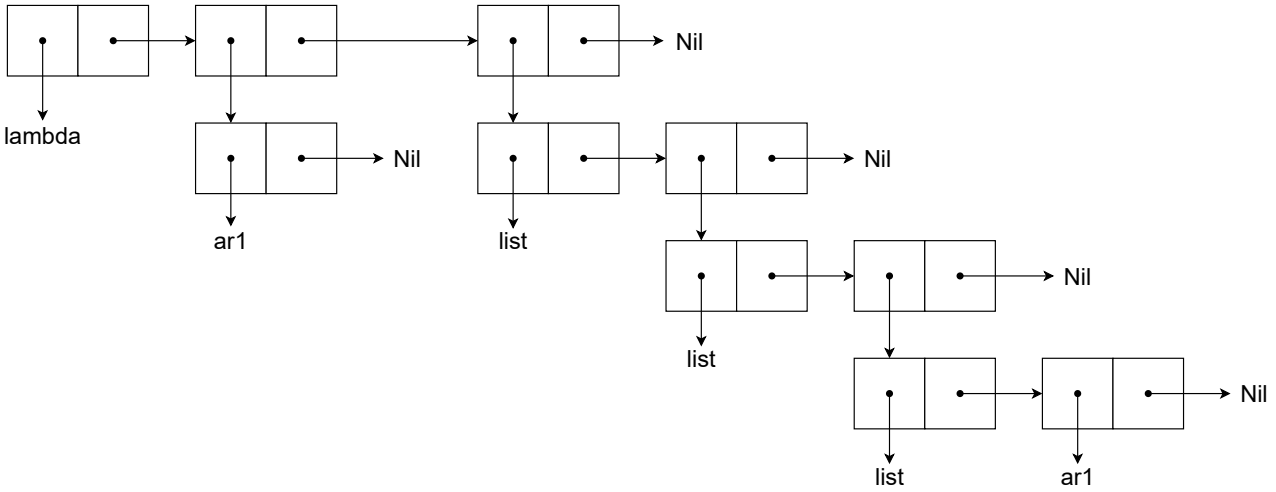

Рисунок 1.7 – Списочные ячейки для листинга 1.14

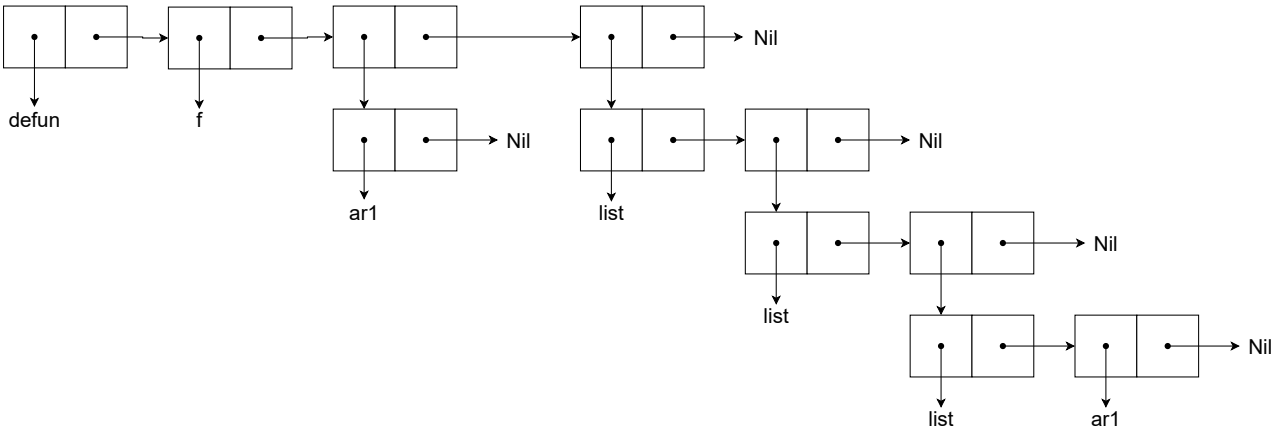

Рисунок 1.8 – Списочные ячейки для листинга 1.15

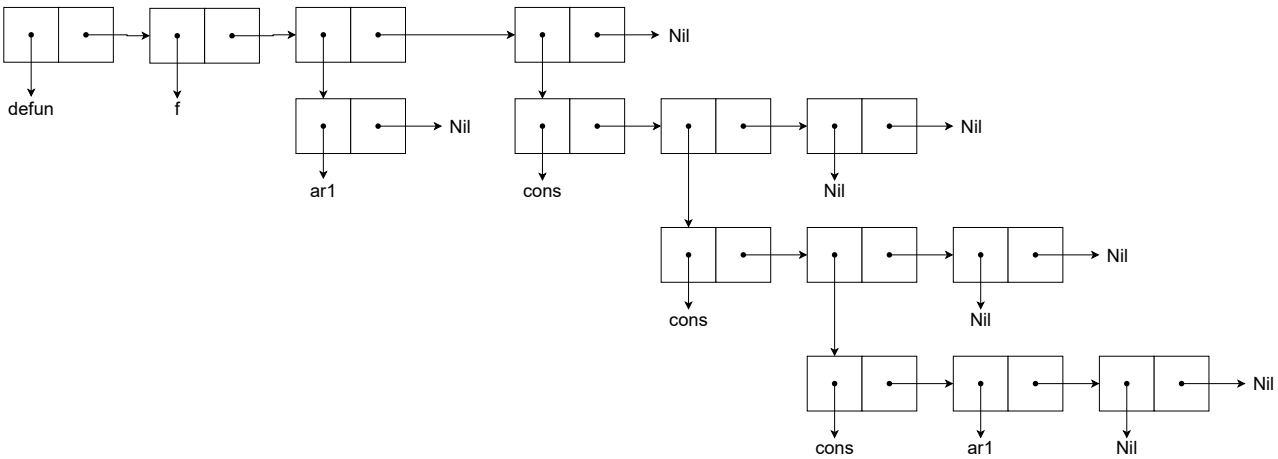

Рисунок 1.9 – Списочные ячейки для листинга 1.16

2 Контрольный вопросы

2.1 Элементы языка: определение, синтаксис, представление в памяти

Определение элементов языка

Элементы языка – атомы и точечные пары (структуры, которые строятся с помощью унифицированных структур - блоков памяти - бинарных узлов). Атомы бывают:

- **символы** (идентификаторы) – синтаксически представляют собой набор литер (последовательность букв и цифр, начинающаяся с буквы; могут быть связанные и несвязанные);
- **специальные символы** – используются для обозначения «логических» констант (`T`, `Nil`);
- **самоопределимые атомы** – числа, строки - последовательность символов в кавычках (`"abc"`).

Синтаксис элементов языка

`S-выражение ::= <атом> | <точечная пара>`
`Точечная пара ::= (<атом> . <атом>)`
`| (<точечная пара> . <атом>)`
`| (<атом> . <точечная пара>)`
`| (<точечная пара> . <точечная пара>)`
`Список ::= <пустой список> | <непустой список>`, где
`<пустой список> ::= () | Nil`,
`<непустой список> ::= (<S-выражение>. <список>)`,
Список – частный случай S-выражения.

Синтаксически любая структура (точечная пара или список) заключается в круглые скобки:

- `(A . B)` – точечная пара;
- `(A)` – список из одного элемента;

- Непустой список – `(A . (B . (C . (D . Nil))))` или `(A B C D)`;
- Пустой список – `Nil` или `()`.

Элементы списка могут быть списками, например – `(A (B C) (D (E)))`. Таким образом, синтаксически наличие скобок является признаком структуры – списка или точечной пары.

Представление элементов языка в памяти

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

2.2 Особенности языка Lisp. Структура программы. Символ апостроф

Особенности языка `Lisp`

К особенностям языка `Lisp` относятся:

- список является базовой конструкцией языка;
- отсутствие типов данных (так как все представляется S-выражением);
- такая трактовка логических значений, при которой истинным считается любое выражение (атом, список), отличное от `Nil`;
- автоматическое динамическое распределение памяти;
- единая синтаксическая форма записи программ и данных, что позволяет обрабатывать структуры данных как программы и модифицировать программы как данные;
- все операции над данными оформляются и записываются как функции, которые имеют значение, даже если их основное предназначение – осуществление некоторого побочного эффекта (например, определение новой функции);

Структура программы

Программы в `Lisp` понимают как применение функции к ее аргументам (вызов функции). Аргументом функции может быть любая форма `Lisp`. Список, первый элемент которого – представление функции, остальные элементы – аргументы функции, – это основная конструкция в `Lisp`-программе.

Формат: (функция аргумент1 аргумент2 ...)

Названия (имена) функций изображаются с помощью атомов (для наглядности можно предпочитать заглавные буквы:
`CONS`, `CAR`, `CDR`, `ATOM`, `EQ`, ...

Все более сложные формы в `Lisp` понимают как применение функции к ее аргументам (вызов функции). Аргументом функции может быть любая форма.

Символ апостроф

Символ апостроф воспринимается как спецальная функция `quote`. Данная функция блокирует вычисления своего единственного аргумента, то есть он воспринимается как константа. При выполнении функции аргумент обрабатывается по общей схеме.

2.3 Базис языка. Ядро языка

Базис языка

Базис состоит из:

1. структуры, атомы;
2. встроенные (примитивные) функции (`atom`, `eq`, `cons`, `car`, `cdr`);
3. специальные функции и функционалы, управляющие обработкой структур, представляющих вычислимые выражения (`quote`, `cond`, `lambda`, `label`, `eval`).

Ядро языка

Ядро языка было создано в 60-х годах прошлого века известным ученым Дж. Маккарти для решения задач обработки символьной информации.

Помимо Базиса языка, ядро включает в себя наиболее часто используемые функции, не входящие в Базис.