Министерство образования Российской Федерации

Московский Государственный Технический Университет им. Н.Э. Баумана

Отчет по лабораторной работе №1

По курсу «Функциональное и логическое

программирование»

Студент Медведев А.В.

Группа ИУ7-62 Преподаватель Толпинская Н.Б.

Москва, 2019

**Как воспринимается символ `?**

Константные значения могут быть любой сложности, включая вычислимые выражения. Чтобы избежать двусмысленности, предлагается константы изображать как результат специальной функции QUOTE, блокирующей вычисление. Представление констант с помощью QOUTE устанавливаетграницу, далее которой вычисление не идет. Использование апострофа (') - просто сокращенное обозначение функции QUOTE.

**Как представляются списки в ОП?**

Список - это точечная пара (S-выражение, атом, список) или пустой список (Nil). Точечные пары состоят из унифицированных структор - блоков памяти- бинарных узлов. Каждый бинарный узел имеет небольшой объем, достаточнй для хранения двух типизированных указзателей (CAR и CDR, левый и правый, голова и хвост)

Пара из первого элемента списка ("голова") и остальных элементов списка ("хвост") представляют собой пару указателей на фтомы

**Как выполняются функции CAR и CDR и какие результаты они вернут в разных случаях?**

Базовыми функциями доступа в данным в lisp являются функции CAR и CDR.

* CAR - В качесве аргумента принимает список (1 аргумент) и возвращает первый элемент списка, если список не пустой и Nil - иначе
* CDR - в качесве аргумента принимает список (1 аргумент) и возвращает список состоящий из всех элементов кроме первого если список не пустой и Nil - иначе

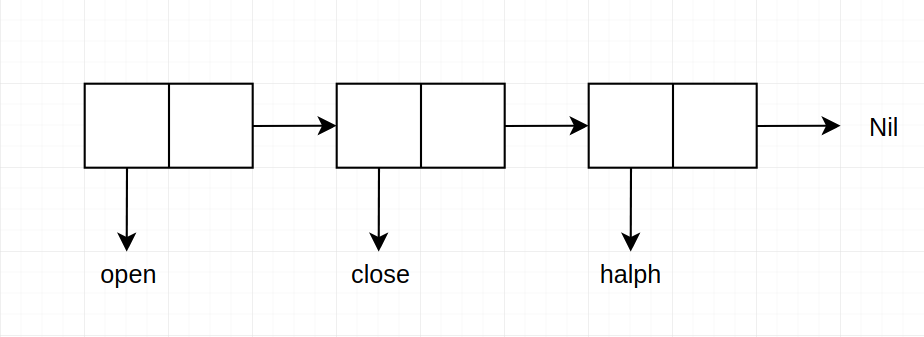
**В чем отличие выполнения функций list и cons?**

Базовоыми функциями создания списков в Lisp явлются функции cons и list.

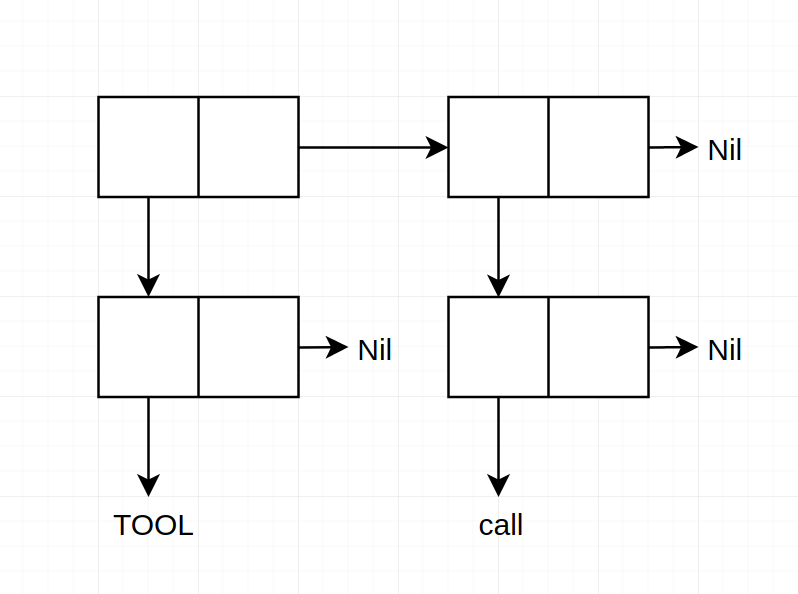
* CONS - в качесве аргумента принимает два аргумента, каждый из которых S-выражение. Возвращает: 1) точечную пару, если второй аргумент не список; 2) Новый список, если второй аргумент список, а первый аргумент добавляет в начало.
* LIST - в качесве аргументов принимает переменное число аргументов (S-выражений). Возвращает список элементами которого являются переданные аргументы. (при отсутвии аргументов вернется пустой список)

1. **Представить следующие списки в виде списочных ячеек:**

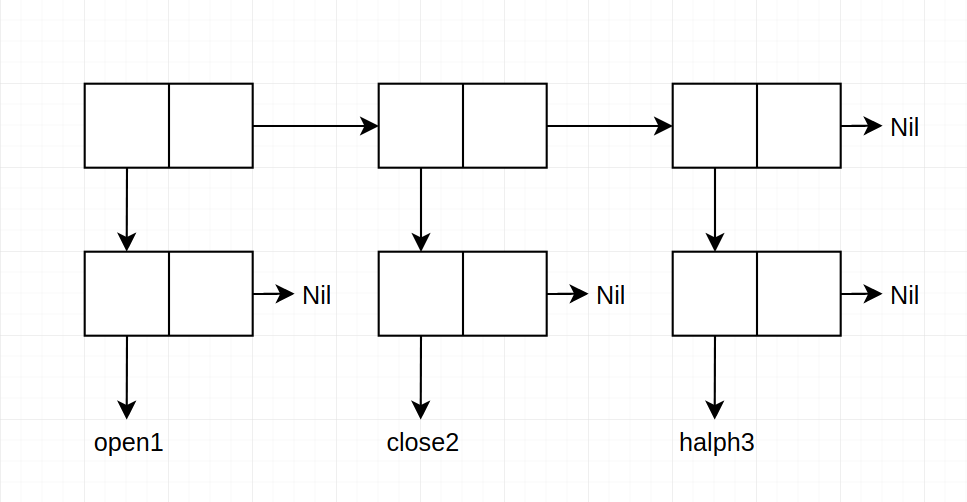
'(open close halph)



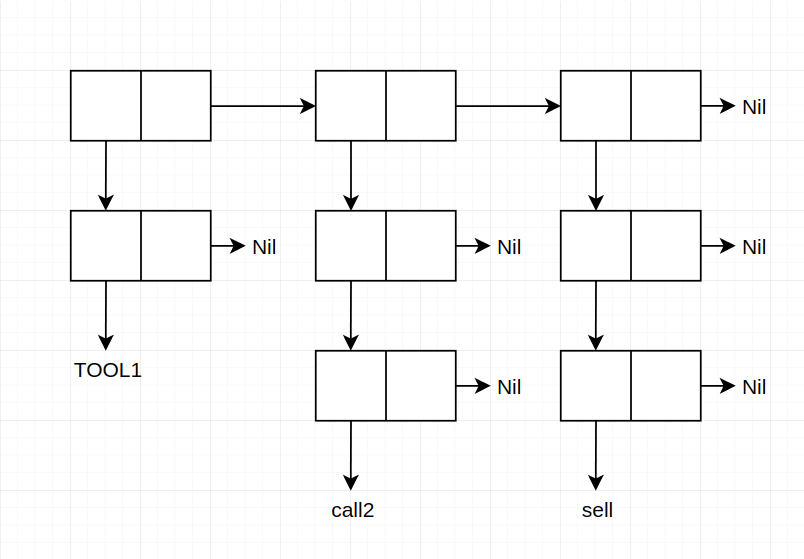
'((TOOL) (call))



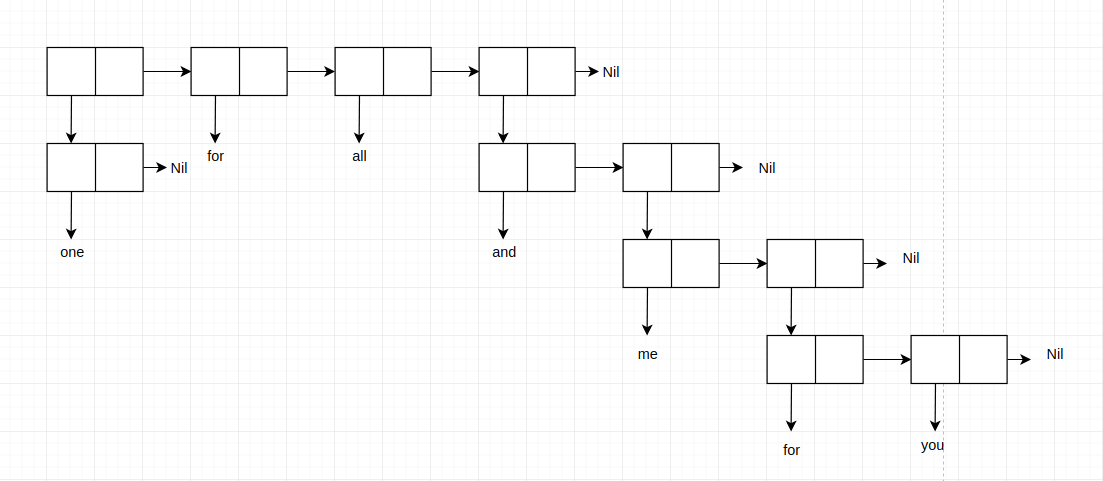
'((open1) (close2) (halph3))



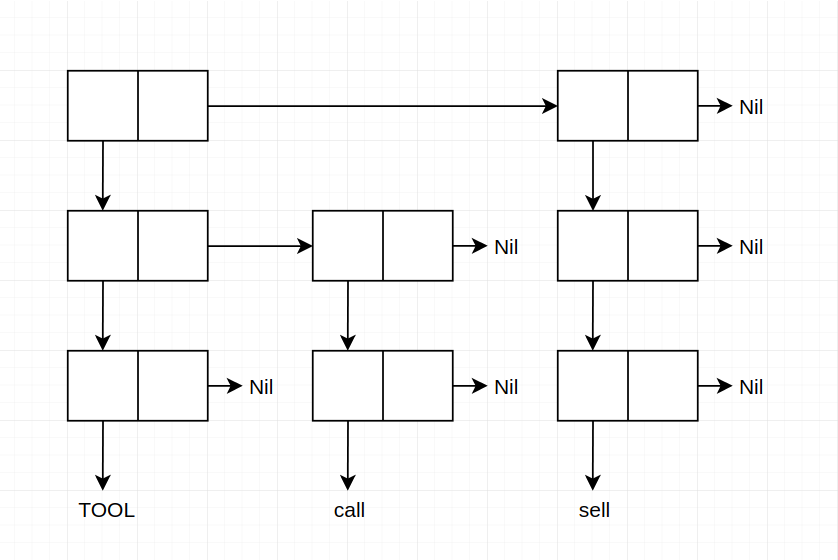
'((TOOL1) ((call2)) ((sell)))



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



'( ((TOOL) (call)) ((sell)) )



**2. Используя только функции car и cdr,написать выражения, возвращающие**

**1) второй**

**2) третий**

**3) четвертый элементы заданного списка.**

|  |  |  |
| --- | --- | --- |
|  | Выражение | Результат |
| Второй элемент списка | (CAR (CDR '(1 2 3 4 5))) | 2 |
| Третий элемент списка | (CAR (CDR (CDR '(1 2 3 4 5)))) | 3 |
| Четвертый элемент списка | (CAR (CDR (CDR (CDR '(1 2 3 4 5))))) | 4 |

**3. Что будет в результате вычисления выражений**

|  |  |  |  |
| --- | --- | --- | --- |
| **Выражение** |  |  | **Результат** |
| **(CAADR '((blue cube) (red pyramid)))** | ((red pyramid)) | (red pyramid) | **red** |
| **(CDAR '((abc) (def) (ghi)))** | (abc) |  | **Nil** |
| **(CADR '((abc) (def) (ghi)))** | ((def) (ghi)) |  | **(def)** |
| **(CADDR '((abc) (def) (ghi)))** | ((def) (ghi)) | ((ghi)) | **(ghi)** |

**4. Напишите результат вычисления выражений:**

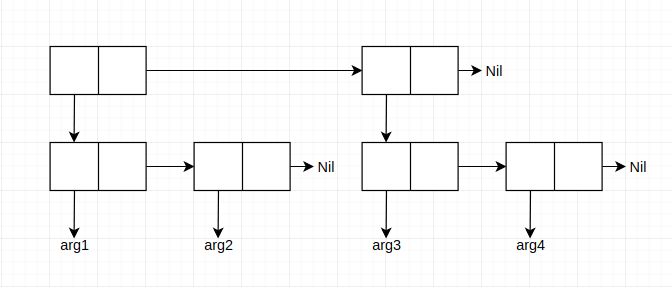
|  |  |
| --- | --- |
| (list 'Fred 'and Wilma) | Wilma unbound |
| (cons 'Fred '(and Wilma)) | (Fred and Wilma) |
| (list 'Fred '(and Wilma)) | (Fred (and Wilma)) |
| (cons 'Fred '(Wilma)) | (Fred Wilma) |
| (cons Nil Nil) | (Nil) |
| (list Nil Nil) | (Nil Nil) |
| (cons T Nil) | (T) |
| (list T Nil) | (T Nil) |
| (cons Nil T) | (Nil . T) |
| (list Nil T) | (Nil T) |
| (list Nil) | (Nil) |
| (cons T (list Nil)) | (T Nil) |
| (cons (T) Nil) | The function T is undefined |
| (list (T) Nil) | The function T is undefined |
| (list '(one two) '(free temp)) | ((one two) (free temp)) |
| (cons '(one two) '(free temp)) | ((one two) free temp) |

**5. Написать функции**

**(f ar1 ar2 ar3 ar4) -> ((ar1 ar2) (ar3 ar4))**

(defun f (ar1 ar2 ar3 ar4) (list (cons ar1 ar2)(cons ar3 ar4)))

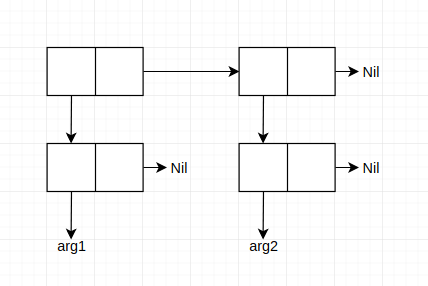
(f 1 2 3 4) - > ((1 2)(3 4))



**(f ar1 ar2) -> ((ar1) (ar2))**

(defun f (ar1 ar2) (list (list ar1)(list ar2)))

(f 1 2) -> ((1)(2))



**(f ar1) -> (((ar1)))**

(defun f (ar1) (list (list (list ar1))))

(f 1) -> (((1)))

