Все теоретические вопросы к лабораторным работам по Lisp в курсе «Функциональное и логическое программирование» (2021 г.)

Оглавление

1. Элементы языка Lisp. Определения. Базовые элементы языка

Вся информация в языке Lisp (и данные, и программа) представляются с помощью символьных выражения, или S-выражений. К S-выражениям относятся атомы и точечные пары.

S-выражение ::= <атом> | <точечная пара>

Основными элементами языка являются S-выражения и списки.

Атомы – элементарные конструкции языка:

- 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, вещественные числа, строки (последовательности символов, заключенных в двойные апострофы).

Точечные пары ::= (<атом>.<атом>) | (<атом>.<точечная пара>) | (<точечная пара>.<атом>) | (<точечная пара >.< точечная пара >).

Точечные пары (структуры) строятся с помощью унифицированных структур – блоков памяти – бинарных узлов.

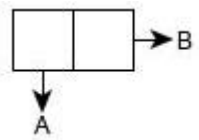
Список – динамическая структура данных, которая может быть пустая или непустая, состоящая из головы и хвоста, который является списком. В Lisp список является частным случаем S-выражения.

2. Синтаксис элементов и их представления в памяти

Любая структура (точечная пара или структура) заключается в круглые скобки.

Точечная пара

(<S-выражение>.<S-выражение>)



Представление в памяти: точечная пара представляется в памяти бинарным узлом.

Пример: (A . B)

Список

Список ::= <пустой список> | <непустой список>, где
<пустой список> ::= ( ) | Nil,
<непустой список> ::= (<голова> . <хвост>),
<голова> ::= <S-выражение>,

<хвост> ::= <список>.
Представление в памяти: одному списку соответствует одна списковая ячейка, которая хранит два указателя на голову (первый элемент) и на хвост (остальной список).

Пример:
(A . (B . (C . (D . (E . Nil)))))
Облеченная форма записи: (A B C D)
Многоуровневый список: (A (B C) (D E))

3. Базис языка Lisp

Базис – это минимальный набор необходимых конструкций, с помощью которого можно реализовать задачу.
В базис Lisp входят:

- атомы и структуры (бинарные узлы)
- базисные функции и функционалы (atom, eq, car, cdr, cons, quote, cond, lambda, label, eval).

4. Особенности Lisp

- как язык функционального программирования ориентирован на символьную обработку;
- и данные, и программа в Lisp представляются в виде символьных выражений – S-выражений (в функциональном программировании данные и программа не различаются).

5. Как воспринимается символ апостроф?

Символ апостроф ‘ – это сокращенная форма записи функции блокировки вычисления QUOTE, которая нужна, чтобы представлять выражения как данные (по умолчанию, выражение воспринимается как программа, где первый элемент списка – название функции, остальные элементы – аргументы функции).

6. Общее понятие рекурсии.

Под рекурсией в общих представлениях подразумевается определение или описание какого-либо объекта или процесса внутри него самого.

7. Классификация функций.

Классификация по «устройству» функций:

- чистые математические: принимают фиксированное число аргументов;
- рекурсивные;
- специальные функции (формы): принимают произвольное число аргументов, которые по-разному обрабатываются;
- псевдофункции: создают эффект на внешних устройствах;
- функции с вариантными значениями, из которых выбирается только одно;
- функции высших порядков (функционалы): используются для синтаксически управляемых программ.

Классификация базисных функций:

- селекторы: car, cdr;
- конструкторы: cons, list;
- предикаты: null, atom, numberp, symbolp, eq, eql, =, equal и т. д.

8. Функции CAR и CDR.

Функции CAR и CDR являются базисными функциями-селекторами, т. е. они осуществляют доступ к элементам списка. Обе функции – чистые математические: они принимают в качестве аргумента точечную пару или список.

Функция CAR переходит по car-указателю к первому элементу списка. В случае пустого списка вернёт Nil.

Функция CDR переходит по cdr-указателю к остальному списку. Если в списке меньше двух элементов, то функция вернёт Nil.

9. Функции CONS и LIST.

Функции CONS и LIST – функции-конструкторы. Обе функции могут использоваться для создания списков.

Функция CONS является базисной, чистой математической функцией. CONS создаёт бинарный узел и устанавливает его указатели на два принятых аргумента.

Примеры:

```
(cons ‘A ‘B) → (A.B)
(cons ‘A ‘(B)) → (A B)
(cons ‘A NIL) → (A)
```

Функция LIST является формой, т. е. принимает переменное число аргументов. LIST возвращает список, элементами которого являются

аргументы функции: car-указатели ссылаются на аргументы, а cdr-указатели «сцепляют» списковые ячейки в список.

Примеры:

```
(list ‘A ‘B) → (A B)
(list ‘A ‘(B)) → (A (B))
(list ‘A NIL) → (A NIL)
```

10. Список, представление и интерпретация списков.

Список – динамическая структура данных, которая может быть пустая или непустая, состоящая из головы и хвоста, который является списком. В Lisp список является частным случаем S-выражения.

Один список представляется одной списковой ячейкой, которая хранит два указателя: car-указатель (на первый элемент, или голову) и cdr-указатель (на остальной список, или хвост).

Первый элемент списка интерпретируется как имя функции, остальные элементы списка – как её аргументы. Если присутствует блокировка вычисления (функция QUOTE, или ‘), то первый элемент также интерпретируется как аргумент.

11. Синтаксическая форма и хранение программы в памяти.

Программа и обрабатываемые ею данных представляются в Lisp одинаково: и то, и другое представляется в виде S-выражения. По этой причине программы могут обрабатывать и преобразовать другие программы и свою собственную.

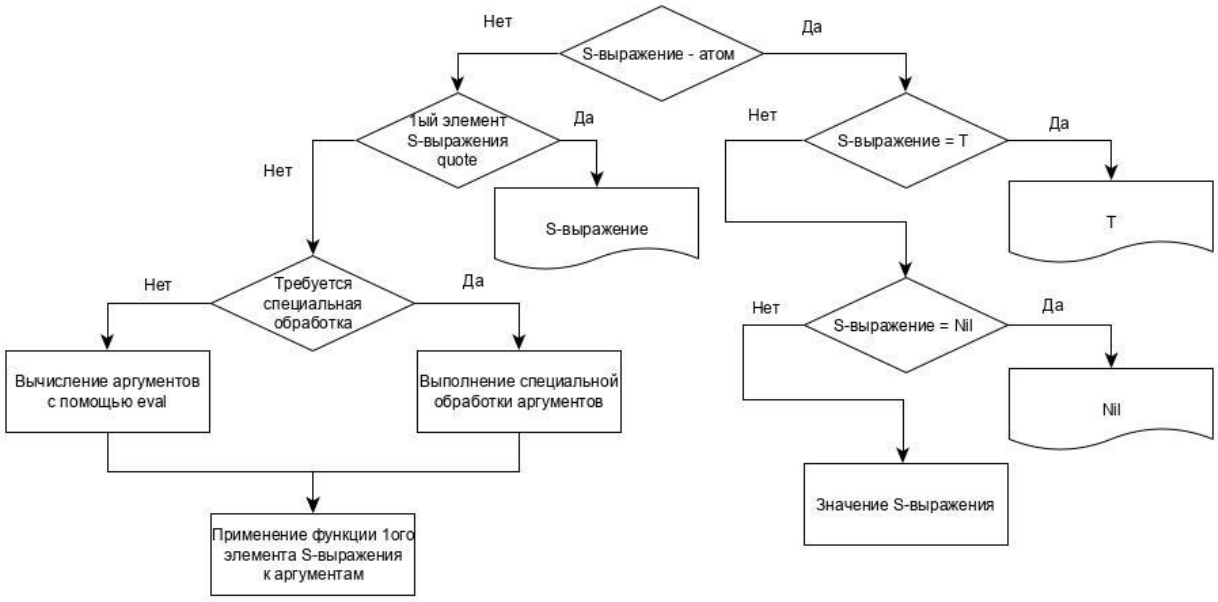
12. Трактовка элементов списка.

Первый элемент трактуется как имя функции, остальные элементы как аргументы. Если стоит блокировка вычисления (функция QUOTE, или в сокращённом виде ’), то первый элемент также трактуется как аргумент.

13. Порядок реализации программы.

Программа работает в цикле:

1. Ожидание ввода S-выражения.
2. Передача введённого S-выражения функции EVAL.
3. Вывод полученного результата.



14. Способы определения функции.

- без имени: `(lambda <лямбда-список> (<тело_функции>))`
- с именем: `(defun <имя> <лямбда-список> <тело_функции>)`

<лямбда-список> - список формальных параметров

15. Работа функций and, or, if, cond.

and – функция-форма, выполняющая логическое умножение («И»).

```
(and <арг1> <арг2> … <аргN>)
```

Функция and проверяет последовательно каждый аргумент на Nil до тех пор, пока не встретит такой аргумент, т. к. результат логического умножения становится известным (Nil). Если все аргументы оказались равными не Nil, то возвращается последний аргумент.

Примеры:

```
(and 'A 'B Nil 'C) -> Nil
(and Nil) -> Nil
(and Nil 'A 'B) -> Nil
(and 'A 'B 'C 'D) -> D
```

or – функция-форма, выполняющая логическое сложения («ИЛИ»).

```
(or <арг1> <арг2> … <аргN>)
```

Функция or проверяет последовательно каждый аргумент на Nil до тех пор, пока не встретит аргумент, который не равен Nil, т. к. результат логического

сложения становится известным (T). Если все аргументы оказались равными Nil, то возвращается последний аргумент, т. е. Nil.

Примеры:

```
(or 'A 'B Nil 'C) -> A
(or Nil) -> Nil
(or Nil 'A 'B) -> A
(or Nil Nil 'C 'D Nil) -> C
```

if – функция-форма, которая возвращает различные значения в зависимости от результата выражения-условия.

```
(if <условие> <T-выражение>[ <F-выражение>])
```

Если выражение «условие» соответствует T (не равно Nil), то вычисляется T-выражение, в противном случае вычисляется F-выражение. F-выражение является необязательным аргументом: если не подаётся этот аргумент, то функция if при невыполнении «условия» возвращает Nil.

cond – функция-форма, которая возвращается различные значения в зависимости от результатов принимаемых выражений-условий.

```
(cond     (test1 value1)
          (test2 value2)
          …
          (testN valueN))
```

Функция cond принимает в качестве аргументов списки, которые содержат в качестве первого элемента предикат, а второго – выражение, которое нужно вычислить, если результат предикат не Nil.

cond вычисляет последовательно предикаты до тех пор, пока не встретится предикат, результат которого не равен Nil. В этом случае вычисляется выражение, соответствующее этому предикату, и результат этого выражения будет возвращён как результат функции cond. Если результат всех предикатов равны Nil, то функция cond возвращает Nil.

16. Разрушающие и неразрушающие структуру списка функции. Варианты и методы модификации списков.

При работе со списками есть два варианта их модификации: с разрушением их структур и без их разрушения.

Разрушающие структуру функции не сохраняют возможность работа со старыми структурами, т. к. эти функции изменяют их.

Неразрушающие структуру списков функции сохраняют такую возможность. Если требуется вернуть модифицированный вариант списка-аргумента, то возвращается его изменённая копия.

Неразрушающие	Разрушающие
append	nconc
reverse	nreverse
last	rplaca
nth	rplacd
nthcdr	
length	
remove	delete
subst	nsubst

17. Отличие в работе функций cons, list, append и в их результатах.

Функция cons – базисная, чистая математическая функция, принимающая ровно два аргумента. Создаёт бинарный узел, расставляя его указатели на два аргумента.

Функция list является формой. Создаёт ровно столько списковых ячеек, сколько передано аргументов. каждый car-указатель ссылается на соответственный аргумент, а cdr-указатель одной ячейки указывает на следующую ячейку.

Функция append является формой, неразрушающей структуру списка: она создаёт новый список, в котором создаются новые списковые ячейки для каждого списка-аргумента, кроме последнего. Cdr-указатель последней списковой ячейки копии одного списка-аргумента указывает на первую списковую ячейку копии следующего списка-аргумента. Cdr-указатель предпоследнего списка будет указывать на первую ячейку самого последнего списка-аргумента, а не его копии.

```
(cons ‘(a b) ‘(c d)) -> ((a b) c d)
(list ‘(a b) ‘(c d)) -> ((a b) (c d))
(append ‘(a b) ‘(c d)) -> (a b c d)
```

18. Порядок работы и варианты использования функционалов.

Виды функционалов и их порядок работы:

- **применяющие**: применяют передаваемую функцию к остальным аргументам
  - `(apply #’func arg_lst) – применяет функцию к аргументам из списка`
  - `(funcall #’func arg1 … argN) – применяет функцию к аргументам arg1…argN`

- **отображающие**: применяют несколько раз передаваемую функцию к аргументам
  - (mapcar #'func '(x1 x2 … xn)) -> ((func x1)(func x2)…(func xn)) – применяют несколько раз передаваемую функцию к аргументам x1…xn
  - (mapcar #'func lst1 lst2 … lstn) – для функций нескольких аргументов: lst1…lstn – списки 1ых, 2ых… Nых аргументов
  - (maplist #'func lst) – применяет несколько раз функцию к аргументам из сначала полного списка, затем из списка без первого элемента, затем без второго и так далее до исчерпания списка.
  - *mapcar ~ mapcan, maplist ~ mapcon: mapcan и mapcon используют nconc при сборке*

  - (find-if #' predicat lst) – поиск первого элемента, удовлетворяющего предикату
  - (remove-if #'predicat lst) – удаляет все элементы, удовлетворяющие предикату
  - (remove-if-not #'predicat lst) – удаляет все элементы, не удовлетворяющие предикату
  - (reduce #'func lst) – каскадно применяет функцию к аргументам списка
  - Пример: (reduce #'+ '(1 2 3 4)) ⇔ (+ (+ (+ 1 2) 3) 4)
  - (every #'predicat lst) – все ли элементы удовлетворяют предикату?
  - (some #'predicat lst) – есть хотя бы один элемент, удовлетворяющий предикату?

Варианты использования функционалов (примеры из лекций):

```
(defun consist-of (lst) (if (member (car lst) (cdr lst) 10)))
(defun all-last-elements (lst)
     (if (eql (consist-of lst) 0) (list (car lst))))
(defun collection-to-set (lst)
     (maplist #'all-last-elements lst))
(collection-to-set '(i t I g t k s i f k)) -> (g t s i f k)
```

```
(defun decart (lstX lstY)
     (mapcan #'(lambda(x)
                (mapcar #'(lambda (y)
                          (list x y)) lstY)) lstX))
```

19. Классификация рекурсивных функций.

Классификация №1:

- простая рекурсия: единственный вызов в теле функции;
- рекурсия первого порядка: вызовов несколько в теле функции;
- взаимная рекурсия: две или несколько функций вызывают друг друга.

Классификация №2:

- хвостовая рекурсия: все необходимые действия выполнены до вызова рекурсии, возвращает результат;
- дополняемая рекурсия: использует дополнительную функцию, которая используется не в аргументе вызова, а вне её;
    - частный случай – cons-дополняемая рекурсия.