Функциональное и логическое программирование

Лекция №1. Теоретические основы функционального программирования

Лекция №2. Базовые функции языка Лисп.

Лисп это древний язык программирования, разработанный Джоном Маккарти в 1961 году. Название языка происходит из «lisp processing» (обработка списков).
Существует альтернативная расшифровка названия LISP: Lots of Irritating Superfluous Parentheses («Много раздражающих лишних скобок») — намёк на особенности синтаксиса языка.

Особенности языка Лисп

1. Символьная обработка
Символьная обработка позволяет эффективно работать с такими структурами, как предложения естественного языка, значения слов и предложений, нечеткие понятия и т. д. И на их основе принимать решения, проводить рассуждения и осуществлять другие, свойственные человеку способы обращения с данными.

2. Одинаковая форма данных и программы
И то и другое представляется списочной структурой, имеющей одинаковую форму. Таким образом, программы могут обрабатывать и преобразовывать другие программы и даже самих себя. Например, можно введенное и сформированное в результате вычислений выражение данных проинтерпретировать в качестве программы и непосредственно выполнить (так называемое программирование, управляемое данными).

3. Хранение данных, не зависящее от места
Списки, представляющие программы и данные, состоят из списочных ячеек, расположение и порядок которых в памяти не существенны.

4. Автоматическое и динамическое управление памятью
Программисту не надо заботиться об учете памяти.

5. Лисп — безтиповой язык программирования
В лиспе имена символов, переменных, списков, функций и других объектов не закреплены предварительно за какими-нибудь типами данных.

Основные структуры данных (символы, числа, списки)

1. Символы
Буквы, цифры, знаки
+ - * / @ $ % ^ & _ > < .(точка) ? ! { } [ ] :

2. Числа
целые числа 56, вещественные 67.89, вещественные в научной нотации 9.1E-31, рациональные числа 5/9.

3. Логические значения
T (истина) NIL (ложь)
любой объект отличный от NIL имеет логическое значение истина.

4. Константы и переменные
Константы это числа и логические значения T и NIL. Остальные символы

5. Атомы
Атомы это символы и числа, плюс логические значения.

6. Списки
Список в Лиспе — упорядоченная последовательность, элементами которой являются атомы и списки (подсписки). Списки заключаются в круглые скобки, элементы списка разделяются пробелами.
Пустой список — список в котором нет ни одного элемента, обозначается () или NIL. NIL может быть элементом списками.

Символьное выражение	примечание
NIL	То же самое, что и ()
(NIL)	Список, состоящий из атома NIL
(())	То же самое, что и (NIL)
((()))	То же самое, что и ((NIL))
(NIL ())	Список из двух пустых списков или двух NIL

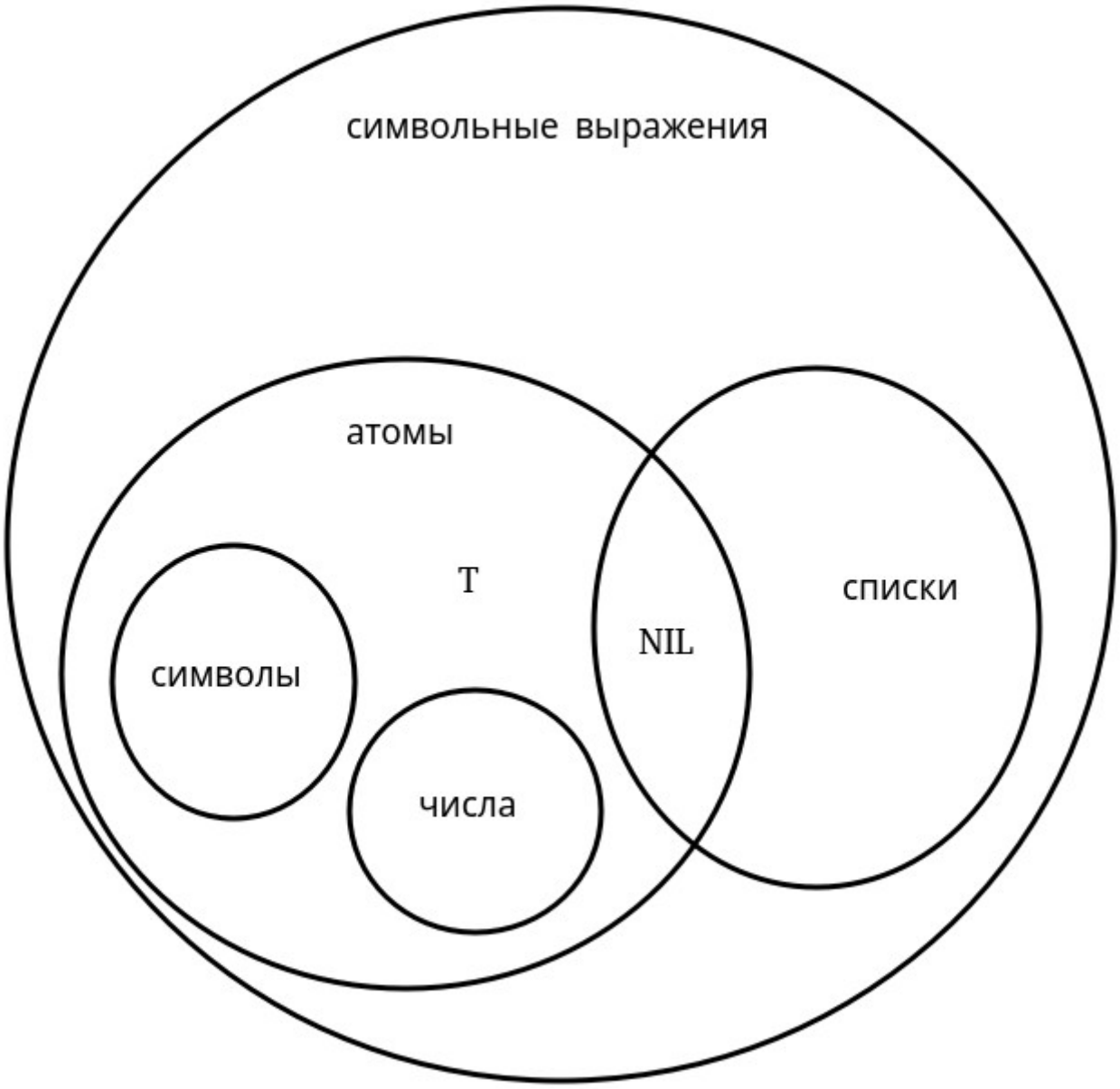

Рис.1 — Символьные выражения.

Список как средство представления знаний, например, информацию о семье можно представить списком с тремя элементами: муж, жена, дети.

```
((husband Tom 45)
 (wife Ann 44)
 (children (Pat woman 22)
           (Bob man 18)))
```

Понятие функции

Мы будем использовать и определять только те фунции Лиспап, которые являются «чистыми», т. е. Не имеют побочных эффектов и значение функции зависит только от

значений её параметров. Т.е. будем учиться «чистому» функциональному программированию.
В отличии от математики и арифметических выражениях, в Лиспе используется единообразная префиксная нотация (табл.2, 3).

Табл. 2 — Примеры записей функций в математике и в Лиспе

Префиксная нотация в математике	**Единообразная префиксная нотация в Лиспе**
f(x)	(f x)
g(x,y)	(g x y)
h(x,g(y z))	(h x (g y z))

Табл.3 — Примеры записей выражений в математике и в Лиспе

Инфиксная нотация в математике	**Единообразная префиксная нотация в Лиспе**
x + y	(+ x y)
x / y	(/ x y)
x * (y + z)	(* x (+ y z))

Блокировка вычислений выражений

В некоторых случаях не надо вычислять значение выражения, а нужно само выражение. Нас, например, может не интересовать значение функционального вызова (+ 2 3), равное 5, а мы хотим обрабатывать форму (+ 2 3) как список. В таких случаях используют специальную функцию QUOTE с одним аргументом. В качестве краткого обозначения для функции QUOTE используют апостроф перед аргументом (табл.4).

Табл.4 — Примеры вызова и результат

Вызов	**Результат**
(quote (+ 2 3))	(+ 2 3)
`(+ 2 `(* 3 5))	(+ 2 (quote (* 3 5)))

Базовые функции

Большинство Лисп-программ можно записать, испоьлзуя только пять простейших функций над символическими выражениями и одну специальную форму (условное выражение):
1. Полиморфный предикат равенства (equal X Y), где X и Y любые символьные выражения, функция возвращает T или NIL.

2. Предикат, проверяющий, является ли символьное выражение X атомом (atom X).
3. Функция (first X) извлекающая первый элемент списка X.
4. Функция (rest X) извлекающая «хвост» списка X.
5. Функция (cons X Y) возвращает список, состоящий из первого элемента X и «хвоста» Y.

Условие (if X Y Z), если X истина, то возвращает Y, иначе Z.

Определение: список это или пустой элемент или последовательность из двух частей: первого элемента и списка.

- - -

Определение: предикат это функция, которая проверяет некоторое условие и возвращает значение логического типа (истина или ложь, T или NIL).

Формат предиката	**действие**
atom <expr>	Проверка, является ли параметр атомом
eq <symb1> <symb2>	Проверка тождественности символов
= <num1> <num2> . . .	Проверка равенства чисел
eql <val1> <val2>	Сравнивает числа одинакового типа, или символы
equal <val1> <val2>	То же что и eql, кроме того, проверка списков
if <expr1> <expr2> <expr3>	условие
null <expr>	Проверка, является ли выражение null
listp <expr>	Проверяет, является ли параметр списком
numberp <expr>	Проверяет, является ли числом
zerop <num>	Выдает инстину, если значение 0 (ноль)
not <expr>	Инвертирует логическое значение выражения
and <expr1> <expr2> . . .	Выдает истину, если все параметры истина
or <expr1> <expr2> . . .	Выдает истину, если хотябы один из параметров истина

Формирование списка из элементов
list <item1> <item2> <item3> . . .

Формат арифметических функций	**действие**
+ <num1> <num2> . . .	сложение
- <num1> <num2> . . .	вычитание
***** <num1> <num2> . . .	умножение
/ <num1> <num2> . . .	деление
sqrt <num>	вычисление квадратного корня
mod <num1> <num2>	остаток от деления
truncate <num>	округление, отбрасывает дробную часть
1+ <num>	Прибавление единицы
1- <num>	вычитание единицы

Определение функций

Программирование на Лиспе сводится к определению необходимых функций, а выполнение программы — к вызову функции с нужными аргументами (фактическими параметрами).
Определение функции:
- дать имя функции
- определить формальные параметры функции
- определить, что должна делать функция (тело функции)
Формат определения функции

```
(defun <имя> (<параметры>)
 <тело функции>
)
```

компонент	синтаксически	смысл
<имя>	атом	Имя функции с помощью которого выполняется вызов функции
<параметры>	список атомов	Список формальных параметров («локальные переменные»), которые имеют значение только внутри функции, инициализируются при вызове, после чего значение поменять нельзя
<тело функции>	символьное выражение	Тело функции, вычисляемое символьное выражение (форма)

Пример: написать функцию вставки символа + между двумя параметрами и формирования списка из этих элементов.

```
(defun insert_plus (s1 s2)
 (list s1 `+ s2)
)
```

Пример вызова:

```
(print (insert_plus 'a 'b))
```

Результат вызова:

```
(A + B)
```

Определение
Форма — символьное выражение, значение которого может быть вычислено интерпретатором:
- константы
- символы, которые используются в качестве переменных
- определения функций и вызовы функций
- специальные формы (quote, if и т. п.)

Пример объявления функции внутри функции

```
(defun insert_plus (s1 s2 s3)
 (defun insert_mul(x1 x2)
```

```
   (list x1 `* x2)
 )
 (insert_mul
  (list s1 `+ s2)
  s3
 )
)
(print (insert_plus 'a 'b 'c))
(print (insert_mul '5 '8))
```
Результат вызова:
```
((A + B) * C)
(5 * 8)
```

т. е. нет смысла объявлять внутри.

Проверка работы функции с несколькими вызовами:
```
(defun test(x1 x2)
  (print "test")
  (+ x1 x2)
  (- x1 x2)
)
(print (test 15 8))
```
результат вызова:
```
"test"
7
```
т. е. работают все вызовы, но результат определяется по последнему вызову.

Некоторые специальные формы

Имя и значение символа.
Символы в Лиспе можно использовать как переменные. В этом случае он могут обозначать некоторые выражения. У символов изначально нет никакого значения как у констант. Попытка вывести значение символа приводит к ошибке.
Пример:
```
(print b)
```
Результат:
```
Error(s), warning(s):
*** - EVAL: variable B has no value
```

При помощи формы **`set`** символу можно присвоить значение.
Формат:
```
(set <expr1> <expr2>)
```
Действие: вычисляется выражение <expr1> если результат является символом, то ему присваивается значение полученное из выражения <expr2>, иначе ошибка.
Пример:
```
(set 'c (* 2 3))
(print c)
```
Результат:
```
6
```

Пример:
```
(defun n2(lst)
  (car (cdr lst)) ;(cadr lst)
)
(set (n2 '(a b c d e)) (+ 2 3))
(print b)
```
Результат:
```
5
```

Форма **setq** тоже самое что и **set**, но 1-й параметр не вычисляется (вычисление блокируется), о чём напоминает буква q в названии функции.
Пример:
```
(setq a `b)
(print a)
```
Результат:
```
B
```

Формы **set** и **setq** имеют побочный эффект — создание связи между символом и значением.
В Лиспе все действия возвращают некоторые значения, в том числе и **set** и **setq** (функции, основное предназначение которых, заключается в осуществлении побочного эффекта). Формы с побочным эффектом в Лиспе можно рассматривать как псевдофункции, так как они всегда возвращают некоторое значение.
Псевдофункции **set** и **setq** представляют оператор присваивания в Лиспе, использование этих псевдофункций нарушает принцип прозрачности по ссылкам. Поскольку мы изучаем чистое функциональное программирование, то формы **set** и **setq** если и будут использоваться, то только для одноразового присваивания значений символам, используемым в качестве глобальных переменных.
Использование псевдофункций **set** и **setq** внутри тела функции несовместимо с чистым функциональным программированием.

Пример использования псевдофункций внутри функций

Объявление функции	**Соответствие парадигме чисто функционального программирования**

`(defun inc1(n)`
`  (setq n (+ n 2))`
`)`
	Не соответствует

`(defun inc2(n)`
`  (+ n 2)`
`)`
	Соответствует

Пример нарушения прозрачности по ссылкам
```
(setq flag T)
(defun f(n)
    (if (setq flag (not flag)) n (* 2 n))
)
```

Вызов	результат

`(print (f 3))`
`(print (f 3))`
`(print (+ (f 1) (f 2)))`
`(print (+ (f 2) (f 1)))`
	`6`
	`3`
	`4`
	`5`

Интерпретатор Лиспа **`eval`**. Вызов этой функции может снять эффект блокировки вычисления или позволяет найти значение выражения (выражение, которое записано как список).

Вызов	результат
`(print (eval '(+ 11 12)))`	`23`