Дополнительный материал 2

Сформулированная Джоном Мак-Карти (1958) концепция символьной обработки информации восходит к идеям Чёрча и других видных математиков конца 20-ых годов предыдущего века. Выбирая лямбда-исчисление как основную модель, Мак-Карти предложил функции рассматривать как общее понятие, к которому могут быть сведены все другие понятия программирования. Функции используют свободные и связанные переменные, реализуемые с помощью таблицы соответствия символов и их толкования. Символьная обработка реализуется путем интерпретации выражений, состоящих из всюду определенных функций, аргументы которых упорядочены.

Функцией называется правило, по которому каждому значению одного или нескольких аргументов ставится в соответствие конкретное значение результата.

Базис Lisp :

- атомы и структуры (представляющиеся бинарными узлами);
- базовые (несколько) функций и функционалов: встроенные — примитивные функции (atom, eq, cons, car, cdr); специальные функции и функционалы (quote, cond, lambda, eval, apply, funcall).

Над базисом возможно **построение формул**, основанное на исчислении Черча:

(ф-я arg1 … argN).

Вычисление – процесс решения задачи, сводимой к обработке чисел, текстов программ или символов, рассматриваемых как модели реальных объектов.

Программы в Лиспе 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 Лисп-программе.

Формат: (функция аргумент1 аргумент2 ...)

Соответствие между моделью и объектом часто называют интерпретацией.

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

Все более сложные формы в Лиспе понимают как применение функции к ее аргументам (вызов функции). Аргументом функции может быть любая форма. Список, первый элемент которого – представление функции, остальные элементы - аргументы функции, – это основная конструкция в Лисп-программе.

Композиции функций естественно строить с использованием скобок.

Формат: (функция1 (функция2 аргумент21 аргумент22 ...) аргумент2 ...)

Построить функцию можно с помощью Lambda-выражения (базисный способ) – Lambda – определения безымянной функции:

```
(LAMBDA (x) (CAR (CDR (CDR x))) )
        | |_____________________|_____определение функции
         | _________________________________ параметр функции
```

При вызове такой безымянной функции необходимо задать фактические значения формальных параметров - связанных переменных:

```
((LAMBDA (x) (atom x)) 12) ; => T
```

X получит значение 12 на время применения построенной безымянной функции, действие которой заключается в выяснении, атом ли аргумент функции.

Список связанных переменных (параметров функции) называется Lambda-списком). Значения они получат при вызове функции.

Соответствие между названием функции и ее определением (именованную функцию) можно задать с помощью специальной функции - конструктора функций DEFUN, первый аргумент которого - имя функции, второй – собственно именуемое определение функции. Формальным результатом DEFUN является ее первый аргумент, который становится объектом другой категории. Он меняет свой статус – теперь это имя новой функции.

```
(DEFUN NEWFUN (x) (CAR (CDR (CDR x))) ))
          |      |  |____________________|________ определение функции
          |      |_____________________________ параметры функции
          |_____________________________________ имя новой функции
```

Новая функция "NEWFUN" действует так же как "Caddr".

Именование функций работает подобно заданию значений “переменным”. Идентификатор представляет структуру, символизирующую функциональный объект. В ней содержится список формальных параметров функции и тело ее определения – аргументы для лямбда-конструктора.

Представления функции могут вычисляться и передаваться как параметры или результаты других функций.

Интерпретация S-выражения, попадающего на вход Lisp-машины, выполняется универсальной функцией. Это функция eval, которая может вычислять значение любой формы Lisp, включая формы, сводимые к вычислению произвольно заданной функции, применяемой к аргументам, представленным в этой же форме, по доступному описанию данной функции.