ЛЕКЦИЯ 3

Основы функционального программирования на языках Lisp, Scheme и FP

ЯЗЫК ЛИСП (LISP) БЫЛ РАЗРАБОТАН В 1958 ГОДУ АМЕРИКАНСКИМ УЧЕНЫМ ДЖОНОМ МАККАРТИ КАК ФУНКЦИОНАЛЬНЫЙ ЯЗЫК, ПРЕДНАЗНАЧЕННЫЙ ДЛЯ ОБРАБОТКИ СПИСКОВ (LIST PROCESSING)

В ОСНОВУ ЯЗЫКА ПОЛОЖЕН СЕРЬЕЗНЫЙ МАТЕМАТИЧЕСКИЙ АППАРАТ:

- ЛЯМБДА-ИСЧИСЛЕНИЕ ЧЕРЧА
- АЛГЕБРА СПИСОЧНЫХ СТРУКТУР
- ТЕОРИЯ РЕКУРСИВНЫХ ФУНКЦИЙ

S-выражение (Simbolic expresion) - основная структура данных в ЛИСПе

(ДЖОН СМИТ 33 ГОДА) \
S-ВЫРАЖЕНИЯ
((МАША 21) (ВАСЯ 24) (ПЕТЯ 1)) /

S-выражение - это либо атом, либо список

АТОМЫ - ЭТО ПРОСТЕЙШИЕ ОБЪЕКТЫ ЛИСПА, ИЗ КОТОРЫХ СТРОЯТСЯ ОСТАЛЬНЫЕ СТРУКТУРЫ.

ПРИМЕР: **ДЖОН AB13 B54 10A**
ПРИМЕР: X, 1997, СИМВОЛ, FUNCTION

АТОМЫ БЫВАЮТ ДВУХ ТИПОВ **- СИМВОЛЬНЫЕ И ЧИСЛОВЫЕ**
СИМВОЛЬНЫЕ АТОМЫ - ПОСЛЕДОВАТЕЛЬНОСТЬ БУКВ И ЦИФР, ПРИ ЭТОМ ДОЛЖЕН БЫТЬ ПО КРАЙНЕЙ МЕРЕ ОДИН СИМВОЛ ОТЛИЧАЮЩИЙ ЕГО ОТ ЧИСЛА. СИМВОЛ КАК ПРАВИЛО ОБОЗНАЧАЕТ КАКОЙ-ЛИБО ПРЕДМЕТ, ОБЪЕКТ ВЕЩЬ, ДЕЙСТВИЕ. СИМВОЛЬНЫЙ АТОМ РАССМАТРИВАЕТСЯ КАК НЕДЕЛИМОЕ ЦЕЛОЕ.
К СИМВОЛЬНЫМ АТОМАМ ПРИМЕНЯЕТСЯ ТОЛЬКО ОДНА ОПЕРАЦИЯ: **СРАВНЕНИЕ**

В СОСТАВ СИМВОЛА МОГУТ ВХОДИТЬ: + - * / @ $ % ^ _ \ <>
ЧИСЛОВЫЕ АТОМЫ - ОБЫКНОВЕННЫЕ ЧИСЛА
124
-344
4.5 3.055E8
ЧИСЛА - ЭТО КОНСТАНТЫ.
ТИПЫ ЧИСЕЛ ЗАВИСЯТ ОТ РЕАЛИЗАЦИИ ЛИСПА
АТОМ - ПРОСТЕЙШЕЕ S-ВЫРАЖЕНИЕ

В ЛИСПЕ СПИСОК - ЭТО ПОСЛЕДОВАТЕЛЬНОСТЬ ЭЛЕМЕНТОВ.

ЭЛЕМЕНТАМИ ЯВЛЯЮТСЯ ***ИЛИ АТОМЫ, ИЛИ СПИСКИ.*** СПИСКИ ЗАКЛЮЧАЮТСЯ В ***СКОБКИ,*** ЭЛЕМЕНТЫ СПИСКА РАЗДЕЛЯЮТСЯ ***ПРОБЕЛАМИ.***

СПИСОК, В КОТОРОМ НЕТ НИ ОДНОГО ЭЛЕМЕНТА, НАЗЫВАЕТСЯ **ПУСТЫМ СПИСКОМ И ОБОЗНАЧАЕТСЯ "()"** ИЛИ СИМВОЛОМ **NIL.**
ПРИМЕРЫ: **(NIL)** ; СПИСОК СОСТОЯЩИЙ ИЗ АТОМА
(NIL ()) ; СПИСОК ИЗ ДВУХ ДРУГИХ СПИСКОВ
NIL ОБОЗНАЧАЕТ КРОМЕ ЭТОГО, В ЛОГИЧЕСКИХ ВЫРАЖЕНИЯХ ***ЛОГИЧЕСКУЮ КОНСТАНТУ "ЛОЖЬ" (FALSE).***
ЛОГИЧЕСКОЕ "ДА"(TRUE) ЗАДАЕТСЯ СИМВОЛОМ "T".

АТОМЫ И СПИСКИ - ЭТО СИМВОЛЬНЫЕ ВЫРАЖЕНИЯ ИЛИ S-ВЫРАЖЕНИЯ
ПРИМЕР СПИСКА: (A(B(C)))
(БАНАН) - 1 АТОМ
(Б А Н А Н) - 5 АТОМОВ
(A. B) – A И B
(B. C) – B И C
(A. (B.C)) (A. (B. (C. (D. NIL))))

Scheme (СКИМ) - диалект языка программирования Лисп. Scheme был разработан Гаем Стилом (Guy L. Steele) и Джеральдом Сассменом (Gerald Jay Sussman).
Основные особенности Scheme:

- минимализм языка, основанный на лямбда-исчислении, которое используется для получения значительной части синтаксиса языка (11 из 23 синтаксических конструкций) из более примитивных конструкций.
- статическая лексическая область видимости: имя переменной относится к самой локальной области видимости; таким образом, код можно читать и интерпретировать вне зависимости от того, в каком контексте он будет вызываться.
- блоки, выражающиеся тремя конструкциями let, let* и letrec.
- "правильная" хвостовая рекурсия, позволяющая записывать итеративные алгоритмы более идиоматично, через рекурсию, и при этом оптимизирующая их так, чтобы поддерживать неограниченное количество активных вызовов.
- продолжения (абстрактные представления состояний программы) как объекты первого класса (процедура call-with-current-continuation).
- общее пространство имен для переменных и процедур. В качестве базовых структур данных язык использует списки и одномерные массивы («векторы»).

С точки зрения синтаксиса Scheme - классический упрощённый язык программирования. Взгляните на простой пример вычисления факториала:

```
(define (factorial n)
(if (= n 0)
1
(* n (factorial (- n 1)))))
```

Кроме порядка записи и большого количества скобок, ничего необычного

Несколько базовых принципов языка Scheme:

1. Круглые скобки. Любое законченное выражение должно быть заключено в них.

2. Никаких дополнительных служебных символов. Хватит скобок. Точка с запятой отделяет от кода комментарии.

3. Построение конструкций по типу «действие-предмет».

Самые распространённые реализации:

Racket - одна из самых полных реализаций, включает в себя удобную обучающую среду Dr.Scheme. Есть версии для платформ Windows, Linux, Mac OS.

Bigloo - тоже достаточно полная реализация. Доступна для платформ Windows, Linux.

LispMe - версия для карманных компьютеров с операционной системой Palm OS.

Gambit-C - один из самых быстрых компиляторов Scheme.

Введение в синтаксис. Сперва познакомимся с несколько необычным порядком слов этого языка: «действие - предмет». Но необычен он только в сравнении с популярными языками программирования. В русском языке такая последовательность нередка:

- Сумма трёх и пяти.
- Произведение пяти, шести и семи.
- Купи в булочной батон

Каждая законченная фраза на этом языке должна быть окружена парой круглых скобок. Запишем сказанное выше на Scheme:

(+3 5)
(*5 6 7)
(купить булочная батон)

Можно записать выражения и посложнее:
(купить булочная батон (2+1))
«Купи в булочной батоны: два плюс ещё один».

FP («FUNCTIONAL PROGRAMMING») - АЛГЕБРАИЧЕСКИЙ ЯЗЫК ПРОГРАММИРОВАНИЯ

1. Атомы. FP использует следующие типы атомов:
- скалярные значения.
- последовательности.
- значение «неопределенность».

2. Функции могут быть элементарными или определенными программистом.

Элементарные функции FP включают следующие:
- тождество: id:x возвращает x.
- константа: %x:y возвращает x для любого значения y.
- математические +,-,*,/: +:<x y> возвращает x+y, если x и y - скалярные значения.
- выборочная функция (селектор): i:<x1 ... xn> возвращает xi, если 1 <= i <= n.
- сравнения =,>,<:=:<x y> возвращает T, если x=y, и F в противном случае.
- ряд векторных и матричных функций: конкатенация, присоединение в начало/конец, циклическая перестановка, транспонирование и т.д.

Определение функции программистом имеет следующий синтаксис: { functionName (function code) }. После определения функция вызывается так же, как элементарные функции.

3. Функциональные формы не могут определяться программистом, т.е. используются только формы, встроенные в язык. Список стандартных форм FP включает в себя следующие:
- составление: [f1, ..., fn]:x эквивалентно <f1:x ... fn:x>.
- композиция: f @ g:x эквивалентно f:(g:x).
- условный выбор записывается как (condition -> trueChoice ; falseChoice) и применяется к атому x по следующему правилу: сначала вычисляется функция condition:x; если ее результат - T, возвращается trueChoice:x, иначе - falseChoice:x. Все три аргумента условного выбора - функции.
- применить ко всем: @f:<x1 ... xn> эквивалентно <f:x1 ... f:xn>.
- правая вставка: !f:<x1 x2 ... xn> эквивалентно f:<x1 !f:<x2 ... xn>> (и применяется рекурсивно); !f:<x> эквивалентно x (левая вставка вычисляется аналогично)