Иевлев Станислав Игоревич
Ведущий разработчик ООО «Альт Линукс».
Окончил математический факультет МПГУ по специальности «Математика и информатика».
Автор статей в области системного программирования и защиты информации.

Ваш новый язык – Scheme

Знакомство с грамматикой

Вы хорошо знаете русский, возможно, неплохо уже говорите по-английски, в школе Вас научили несколько необычному языку математики. Предлагаю Вам выучить ещё один – LISP. Точнее, один из его самых интересных диалектов – Scheme (правильно произносится название как «ским», однако можно говорить и проще – «схема»).

Начнём с того, что ознакомимся с не-сколько необычным порядком слов этого языка: «действие объект объект ...». Хотя почему необычным? Вот несколько фраз на русском языке, организованных по тому же принципу:
«Сумма трёх и пяти»,
«Произведение пяти, шести и семи».
Каждая законченная фраза на этом языке должна быть окружена парой круглых скобок.

Запишем сказанное выше на Scheme:

```
(+ 3 5)
(* 5 6 7)
```

Можно записать выражения и посложнее:
`(+ 2 3 (+ 5 6) )` – «Сумма двух, трёх и суммы пяти и шести.»
Просто, не правда ли?

Давайте двигаться дальше.
Фраза «`(* 3 5)`» – хороша, а «`(* width height)`» лучше. Выражение «`(* 2 3.1415926 5)`» – интригующе, а «`(* 2 pi radius)`» гораздо более осмысленно. Здесь width, height – переменные, а 3, 5 – их текущие значения.

Переменная задаётся следующей кон-струкцией языка:
(define имя «первоначальное значение»).
Пример:

```
(define width 3)
(define height 7)
(* 2 (+ width height))
```

Прочитаем записанное по-русски: «Поло-жим ширина – это 3, высота – это 7, подсчитаем произведение двух и суммы ширины и высоты (например, периметр прямоугольника)». Результат такого вы-числения в нашем случае будет 20.

Продолжим совершенствовать конст-рукции. Допустим, нам требуется подсчи-тать сумму квадратов двух чисел. Это можно сделать, например, так:

```
(define a 3)
(define b 4)
(+ (* a a) (* b b))
```

Не правда ли, что-то режет глаз? Слишком громоздко – так мы никогда не говорим. Если бы у нас был в языке глагол, который означал бы «квадрат числа», то последнее выражение звучало бы:

```
(+ (square a) (square b))
```

Согласитесь, что так гораздо лучше.

Есть задача – есть её решение. Мы можем объявить новое слово-функцию, назвать её square. Функция будет принимать в качестве параметра число и возвращать его квадрат. Делается это следующим образом:

```
(define (square x) (* x x))
```

Общий формат:

(define (название параметр параметр...) тело функции)

Функция возвращает последнее вычислен-ное значение. Это означает, что следую-щая функция square2:

```
(define (square2 x) (+ x x) (* x x))
```

вернёт тот же результат, что и square, только попутно сделает ещё одно ненуж-ное дело – удвоение числа.

Перепишем пример с суммой квадра-тов чисел заново:

```
(define a 3)
(define b 4)
(define (square x) (* x x))
(+ (square a) (square b))
```

Нам не хватало слов в языке – мы их добавили. Вообще, когда вы пишете про-грамму на LISP, вы описываете не алго-ритм, а сначала создаёте язык, а потом на нём формулируете исходную задачу. Несколько точнее – вы «подгоняете» данный вам Scheme язык до тех пор, пока он не станет совпадать с языком, на котором задача формулируется легко.

Сразу пример. Пусть перед нами стоит задача сделать программу, которая спра-шивает имя пользователя, а потом выводит ему приветствие.

Scheme предоставляет нам несколько готовых «глаголов»:

- read – для чтения имени,
- display – для того чтобы рисовать что-то на экране,
- newline – для того чтобы рисовать на экране «перевод строки».

Мы хотели бы иметь такие «глаголы»:

- hello – для приветствия с одним параметром – именем пользователя,
- username – для получения имени пользователя, без параметров.

На таком языке наша задача выглядела бы следующим образом:

```
(hello (username))
```

Дело за малым – определить hello и username. Нет проблем. Вот полный текст программы:

```
(define (hello name)
  (display "Hello ")
  (display name)
  (display "!")
  (newline))
(define (username)
  (write "Enter your name:")
  (read))
(hello (username))
```

LISP – полноценный функциональный язык, а поэтому функции – полноправные члены этого языка, независимо от того, определили вы их сами, или они уже были в языке «готовые». В частности, их можно передавать в качестве параметров в другие функции, а там уже делать с ними всё, что потребуется.

Например, функцию «модуль числа» мож-но определить так:

```
(define (abs x)
  (if (positive? x )
      x
      (- x)))
```

«Если аргумент положителен – возвраща-ется x, иначе – минус x». А можно и так:

```
(define (abs x)
  ((if (positive? x) + -) x))
```

«Если аргумент положителен, то плюс, иначе – минус x». Здесь в результате исполнения выражения `if` возвращается функция + или -, которая затем применяется к аргументу x. Кстати, обратите внимание на новую конструкцию `if` – полагаю, что её назначение вам сразу ясно. Сначала проверяется первый аргумент, если он истинен, то исполняется второй аргумент, иначе – третий. Общий формат следующий:

```
(if условие «действие, если условие выполняется» «действие в противном случае»).
```

Где посмотреть и попробовать

В теории всё хорошо, а где немного попрактиковаться? В мире можно найти много прекрасно разработанных сред для работы со Scheme. К сожалению, большинство документации по Scheme на английском языке, но можно найти и отличные сведения на русском – язык-то простой.

Вот названия нескольких самых распространённых реализаций:

Plt Scheme(http://www.plt-scheme.org/) – одна из самых полных реализаций, вклю-чает в себя удобную обучающую среду Dr.Scheme. Есть версии для платформ Windows, Linux, Mac OS.

Bigloo (http://www-sop.inria.fr/mimosa/fp/Bigloo/) – тоже достаточно полная реализация. Доступна для платформ Windows, Linux.

LispMe (http://www.lispme.de/index.html) – версия для карманных компьютеров с операционной системой Palm OS.

Также ещё посмотрите Gambit-C (http://www.iro.umontreal.ca/~gambit/) – один из самых быстрых компиляторов Scheme.

Все перечисленные реализации Scheme – это интерпретаторы. Запускаете интерпретатор – и можно вести с ним диалог на Scheme: в ответ на его приглашение вводите конструкции на Scheme, а он будет возвращать результаты вычислений.

```
Wecome to MzScheme version 208,
Copyright (c) 2004  PLT Scheme,
Inc.
>1
1
>(+ 1 2)
3
>(define a 3)
>(+ a a)
6
>
```

Попробуйте «проиграть» все вышепере-численные примеры. Думаю, Вам понра-вится!

Кто в мешке

Простота Scheme обманчива. На самом деле – это один из самых мощных на сегодняшний день языков программирования. На основе этого языка можно изучить все известные стили и методы программи рования. С частью этих приёмов мы познакомимся с вами в следующих статьях этой серии.

Упражнение. Посмотрите следующие две реализации функции расчёта факториала $f(n)=1 \cdot 2 \cdot \ldots \cdot n$. Одна из них основана на рекурсии, а другая – на итерациях. Напишите на языке Scheme рекурсивную и итеративную реализации функции возведения в степень $f(a, n) = a^n$.

Вариант1

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

Вариант2

```
(define (fact-iter result counter)
 (if (= counter 0)
     result
     (fact-iter (* counter result)
                (- counter 1))))
(define (factorial n) (fact-iter 1 n))
```