Встроенные типы данных языка Scheme

Коновалов А. В.

27 сентября 2022 г.

Литерный тип (character)

Слово «символ» в русском языке, применительно к типам в ЯП, двузначно: это и печатные знаки (из которых состоят строки), и некоторые имена (например, часть компилятора — таблица символов (symbol table) хранит в себе свойства именованных сущностей — переменных, функций, типов и т.д.).

По-английски первое называется «character», второе — «symbol». Чтобы нам не путаться, слово «character» в курсе «Основы информатики» мы будем называть *литерой.*

Литерный тип хранит в себе печатные знаки, т.е. знаки, которые вводятся с клавиатуры и выводятся на экран. Из литер состоят строки.

Сообщение о языке R5RS не описывает множество символов, реализация Racket допускает использование всех знаков Юникода (в том числе, кириллицы).

Предикат типа — `(char? ch)`.

Литералы, т.е. то, как записываются символы в программе. Они бывают двух видов: когда символ можно представить печатным знаком, и когда нельзя. В первом случае они записываются как `#\x`, где `x` — сам знак. Например: `#\a` — строчная латинская `a`. `#\!` — восклицательный знак.

Во втором случае они записываются как `#\‹слово›`, например:

- `#\tab` — знак табуляции
- `#\space` — пробел
- `#\newline` — перевод строки (в Си — `\n`)
- `#\return` — возврат каретки (в Си — `\r`)

Замечу, что `#\n` — строчная латинская буква `n`.

Преобразование между символом и его числовым кодом:

```
(char→integer char)                    → code
(integer→char code)                    → char

(char→integer #\@)                     → 48
(integer→char 48)                      → #\@
```

Сравнение символов (по числовым кодам):

```
(char<? ch1 ch2)
(char>? ch1 ch2)
(char⩽? ch1 ch2)
(char⩾? ch1 ch2)
(char=? ch1 ch2)
```

Сравнение без учёта регистра:

```
(char-ci<? ch1 ch2)
(char-ci>? ch1 ch2)
 ...
```

Преобразование регистра (??):

```
(char-upcase #\a)          → #\A
(char-downcase #\Q)        → #\q
(char-upcase #\1)          → #\1
(char-downcase #\!)        → #\!
```

Предикаты видов литер:

```
(char-whitespace? ch)    ; пробельный символ: пробел, табуляц
                         ; перевод строки и т.д.
(char-numeric? ch)       ; цифра
(char-alphabetic? ch)    ; буква
(char-upper-case? ch)    ; большая буква
(char-lower-case? ch)    ; строчная буква
```

Строковой тип (string)

Строка — последовательность литер. Строка может быть пустой.

Литерал — текст, записанный в двойных кавычках. Внутри строки допустимы стандартные escape-последовательности языка Си. Примеры:

```
"Hello!"
"First line\nSecond line"
"Он крикнул: \"Превед!\""
```

Создание строк:

```
(make-string count char)
(make-string count)
```

Если символ `char` не указан, то создаётся строка, состоящая `count` символов с кодом `0` (т.е. `(integer→char 0)`).

```
(string-ref str k)                    → k-й символ строки
(string-set! str k char)              ; присваивает k-му символу
```

Нумерация ведётся с нуля.

Функция `string-set!` будет работать со строками, созданными при помощи `make-string`, но при этом может не работать со строками, созданными при помощи литералов.

```
(define s1 (make-string 3 #\a))
s1                                    → "aaa"
(define s2 "aaa")
s2                                    → "aaa"
(string-set! s1 1 #\b)
s1                                    → "aba"
(string-set! s2)                      → ОШИБКА
```

Строку можно создать из отдельных литер:

```
(string #\H #\e #\l #\l #\o)        → "Hello"
```

Можно преобразовать строку в список литер и наоборот:

```
(string→list "Hello")                → (#\H #\e #\l #\l #\o)
(list→string '(#\H #\e #\l #\l #\o))   → "Hello"
```

Сравнение строк (в лексикографическом порядке), с учётом регистра и без него (с суффиксом `-ci`):

```
(string=? str1 str2)
(string<? str1 str2)
 ...
(string-ci=? str1 str2)
(string-ci<? str1 str2)
```

(Для всех пяти знаков =, <, >, ⩽, ⩾.)

```
(string<? "Hello" "Hi")          → #t     ; #\e < #\i
(string<? "Hello" "Hell")        → #f     ; вторая строка к
```

Длина строки:

```
(string-length "abcdef")            → 6
```

Выбор подстроки:

```
(substring "abcdef" 2 5)            → "cde"
(substring "abcdef" 2 3)            → "c"
```

Числовые типы

Башня числовых типов (каждый верхний предикат включает в себя все нижние):

```
(number? x)                                  ; это число
(complex? x)                                 ; комплексное число
(real? x)                                    ; вещественное число
(rational? x)                                ; дробное число
(integer? x)                                 ; целое число
```

Литералы типов:

```
3/4                                          ; дробное число
+3.14  6.022e23   1.38e-23                   ; вещественные числа
3+5i   -10-7.5i   2/3+3/4i                   ; комплексные числа
```

Целые числа в Scheme имеют неограниченную точность, т.е. число цифр в них ограничено только памятью компьютера. Внутренне, скорее всего, небольшие числа представлены как длинные машинные числа (т.е. `long` в языке Си), большие числа — как массивы цифр в некоторой системе счисления (например, как `unsinged int[]` в системе по основанию 2^{32}).

Предикат `eq?` может различать два больших равных по значению целых числа, если они в памяти представлены двумя разными массивами.

Вещественные числа имеют ограниченную точность (мантисса имеет конечное число значимых цифр). Скорее всего, они будут представлены как `double`.

Язык Scheme — один из редких языков программирования, где поддерживаются рациональные числа на уровне языка:

```
(/ 1 3)                    → 1/3
(/ 10 3)                   → 3 1/3
```

В Scheme числа делятся на точные (exact) и неточные (inexact). Синтаксически неточные записываются с использованием знаков `.` (точка) и `e` (показатель степени).

Арифметические операции с точными числами дают точный ответ. Если хотя бы один из операндов неточный — результат будет неточный (приближённый).

Точные числа — целые числа, рациональные числа и комплексные числа, обе компоненты которых тоже точные (т.е. целые или рациональные).

Неточные числа — вещественные числа или комплексные с вещественными компонентами.

Предикаты:

```
(exact? num)
(inexcat? num)
```

Есть операции преобразования:

```
(exact→inexact num)          → ближайшее вещественное чи
(inexact→exact num)          → ближайшее дробное число
```

Примеры:

```
(define pi (* 4 (atan 1)))
pi                                  → 3.141592653589793
(exact? pi)                         → #f
(inexact? pi)                       → #t
(inexact→exact pi)                  → 3 398547888715872814749
(define googol 10000000000000000000000000000000000000000000000
googol                              → 10000000000000000000000000
(exact? googol)                     → #t
(exact→inexact googol)              → 1e+100

(exact? 1+2/3i)                     → #t
(exact? 1.0+2i)                     → #f
```

Тип данных `vector`

Списки — основные структуры данных в языках семейства Lisp. В Scheme они не примитивный тип, а надстройка над cons-ячейками.

Списки по своей сути однонаправленны — можем их читать слева-направо при помощи `car` и `cdr` и наращивать справа-налево при помощи `cons`.

Недостаток списков — это производительность при доступе по номеру.

Есть встроенная функция `(list-ref xs n)`, возвращающая n-й элемент:

```
(define xs '(a b c d))
(list-ref xs 2)                    →  c
```

(элементы нумеруются с нуля)

Но сложность той же `list-ref` — `O(n)`, где `n` — номер элемента.

Для преодоления этого недостатка в Scheme есть встроенный тип данных `vector`, допускающий произвольный доступ к элементам для чтения и записи за константное время.

Нужно помнить, что `vector` — ссылочный тип, в том смысле, что если мы в две переменные положим один и тот же вектор, то изменения вектора через одну переменную будут видны через другую.

```
(define v #(1 2 3 4))
(define w v)

(vector-set! v 2 77)
w                                   → #(1 2 77 4)
```

Создаётся вектор при помощи литерала #(…), при этом его содержимое неявно цитируется, также как и при '(…).

```
(define a 100)
(define v #(1 2 3 a 4 5 6))
a                                   → #(1 2 3 a 4 5 6)
```

Т.е. a внутри вектора будет не переменной, а процитированным символом.

Функция `make-vector` создаёт новый вектор:

```
(make-vector size)
(make-vector size init)
```

где `size` — размер вектора, а `init` начальное значение элементов. Т.к. вектора используются чаще для расчётов, инициализация по умолчанию — `0`.

```
(make-vector 10)          → #(0 0 0 0 0 0 0 0 0 0)
(make-vector 10 'a)       → #(a a a a a a a a a a)
```

Предикат типа — `vector?`.

```
(vector? #(1 2 3))                    → #t
(vector? '(1 2 3))                    → #f
```

Обращения к элементам вектора:

```
(vector-ref v n)                    → n-й элемент вектора (нач
(vector-set! v n x)                 ; присваивает n-му элемент
                                    ; новое значение x


(define v (make-vector 5))
v                                   → #(0 0 0 0 0)
(vector-ref v 2)                    → 0
(vector-set! v 2 100)
v                                   → #(0 0 100 0 0)
(vector-ref v 2)                    → 100
```

Что будет?

```
(define m (make-vector 4 (make-vector 4)))
```

На первый взгляд, мы создаём квадратную матрицу. На самом деле, мы создаём два вектора, все элементы одного вектора содержат `0`, все элементы второго — ссылку на первый.

```
m                  → #(#(0 0 0 0) #(0 0 0 0) #(0 0 0 0) #(0 0 0 0
(vector-set! (vector-ref m 0) 0 1)
m                  → #(#(1 0 0 0) #(1 0 0 0) #(1 0 0 0) #(1 0 0 0
```

Вектор можно преобразовать в список и наоборот

```
(vector→list #(a b c))              → (a b c)
(list→vector '(a b c))              → #(a b c)
```

Строки (ещё раз)

Тип данных `string` хранит в себе последовательность литер (characters). Литерал для строки — текст, записанный внутри двойных кавычек: `"Hello!"`.

Внутри строк допустимы стандартные escape-последовательности языка Си:

```
"one line\ntwo lines"                    ; строка со знаком перевод
"I say: \"Hello!\""                      ; заэкранированная кавычка
```

Операции над строками:

```
(make-string 10 #\a)                  → "aaaaaaaaaa"
(string-ref "abcdef" 3)               → #\d                  ; счёт
(string→list "abcdef")                → (#\a #\b #\c #\d #\e #\f
(list→string '(#\H #\e #\l #\l #\o)) → "Hello"
(string? "hello")                     → #t
(string? 'hello)                      → #f
(string-append "штука" "турка")       → "штукатурка"
```

Литеры задаются так:

```
#\x              ; буква «икс»
#\7              ; цифра «семь»
#\(              ; литера «круглая скобка»
#\space          ; пробел
#\newline        ; \n в Си
#\return         ; \r в Си
#\tab            ; \t в Си
#\               ; хотели пробел, но получили ошибку синтаксиса
```

В ДЗ потребуется функция `(whitespace? char)`, возвращающая истину, если литера — пробельная (пробел, табуляция, новая строка, возврат каретки).

```
(whitespace? #\space)                → #t
(whitespace? #\z)                    → #f
(whitespace? (string-ref "a b" 1)) → #t
```

Выбор подстрок `(substring …)` изучить самостоятельно.

Функции преобразования типов

```
(string→number "10.3e7")          → 103000000.0
(number→string 100500)            → "100500"
(string→list "abc")               → (#\a #\b #\c)
(list→vector '(1 2 3))            → #(1 2 3)
(string→number "qwerty")          → #f
```